MASTER EN SCIENCES INFORMATIQUES

Animations fractales d'IFS de forme végétale

LAURENT, Frédéric

Award date:
1993

Awarding institution:
Universite de Namur

Animations fractales d'IFS de forme végétale

Frédéric LAURENT Septembre 1993

Promoteur :
Monsieur Jean FICHEFET

Mémoire présenté en vue de
l'obtention du titre de Licencié
et Maître en Informatique

Année 1992 - 1993

Je tiens à remercier tout particulièrement M. Jean FICHEFET pour avoir accepter la direction de mon mémoire. De plus, son aide m'a permis d'aboutir au présent résultat.

Je remercie également Jean-Benoit PIROT pour m'avoir épaulé dans la réalisation des applications. Sa franche collaboration fut d'un grand secours pour l'aboutissement de ce mémoire.

Enfin, je remercie toute personne ayant collaboré de près ou de loin à la réalisation de ce travail.

Table des matières

Résumé

Certaines théories mathématiques nous offrent la possibilité de réaliser de belles images, de belles représentations d'objets imaginaires. Certains de ces objets tentent d'imiter ce que la nature invente chaque jour. Les objets fractals font partie de ces formes issues des mathématiques et dont le but initial est de modéliser les formes naturelles. L'objectif de ce travail est, non pas la représentation de ces formes et images fantastiques que sont les fractales, mais la réalisation d'une application permettant de leur donner vie, de les mouvoir et de leur apporter une dimension réaliste supplémentaire : **l'animation**. Pour y arriver, nous allons parcourir la théorie des fractales mais également certaines notions utiles sur les séquences animées et l'animation en général. Nous pourrons ensuite mettre en pratique ces aperçus théoriques afin de réaliser quelques programmes d'animations fractales. Signalons enfin que la limitation de ce travail aux objets fractals proches de la botanique vient du fait qu'ils représentent les formes les plus réalistes à animer et offrant la plus grande diversité de mouvements possibles.

Abstract

Some mathematical theories give us the opportunity to realize wonderful representations of imaginary objects. Some of these objects try to imitate the inventions of the nature. The fractal objects are some of these mathematical objects that imitate natural forms. The purpose of this work is to make an application that can animate these fractals and give a rather good representation of their movement. To reach this, we will study a part of the fractal theory and some elements of animated sequences and animation. After that, we will realize three programmes that calculate some fractal animations. The limitation of this work to fractals that represent plants, farns and so on, is due to a greater realism of these objects and a greater diversity of possible movements.

ERRATUM

Une erreur due à un problème d'impression sur une imprante couleur s'est glissée dans les pages 38, 39 et 40. A la page 38, les formules de projections concernant les angles sont les suivantes :

$$X' = x * \cos(\alpha) + y * \cos(\beta) + z * \cos(\gamma)$$
$$Y' = x * \sin(\alpha) + y * \sin(\beta) + z * \sin(\gamma)$$

De plus dans les pages suivantes, à chaque dénomination d'angle, il faut lire α au lieu de ♋, β au lieu de ♌ et γ au lieu de ♑.

Introduction

Comme il l'a été souvent démontré, la théorie fractale est un puissant outil pour la modélisation et la représentation de structures rencontrées dans la nature. Ainsi, cet outil permet la production d'images contenant des nuages, de la fumée, des reliefs montagneux, des flammes, des cristaux de glaces ou encore des fougères, des feuilles d'érables, des arbres... Cette modélisation étant très réaliste, elle permet d'obtenir des images de très bonne qualité avec des objets en deux ou trois dimensions et le tout en couleur. Cependant, une telle représentation reste une image, certes de qualité, mais qui est fixe et sans *'vie'*.

Toujours dans le même souci de représentation de structures naturelles réalistes, il serait bon d'imaginer des algorithmes réalisant l'animation de ces mêmes objets fractals. En effet, ceci donnerait vie à ces structures et leur apporterait la dimension qui leur manque. Ces algorithmes pourraient se baser sur les propriétés de continuité des paramètres de l'objet fractal.

Ce mémoire se propose de tenter une approche de ce problème d'animation d'images fractales. L'étude va se porter sur les objets fractals de type plantes, feuilles, arbres... Cette orientation botanique a pour origine l'aspect réaliste de l'animation d'une plante et la variété des mouvements possibles. Il n'en est pas de même pour une chaîne de montagnes, des nuages ou des cristaux de glaces.

Pour arriver à cet objectif, nous allons d'abord rappeler les bases de la théorie fractale et les éléments qui nous seront nécessaires. Après cela, il serait intéressant d'analyser les principes élémentaires d'une animation et du mouvement d'un objet. Ceci nous permettra de définir notre stratégie pour la réalisation ultérieure des programmes.

Dans un second temps, nous étudierons dans la partie pratique comment réaliser ces programmes d'animations fractales. Cette conception de programmes sous Dos va faire ressortir quelques difficultés et nous conduira à concevoir une application réalisée sous l'environnement Windows et qui sera l'application finale de ce mémoire. Avant d'en arriver aux conclusions à tirer de cette étude, nous étudierons la manipulation de ce programme sous Windows, ce qu'il permet de réaliser et nous proposerons quelques exemples de résultats obtenus.

Mais n'anticipons pas et passons dès à présent aux éléments théoriques qui seront nécessaires à l'élaboration des programmes.

PARTIE THEORIQUE

Notions d'IFS et de séquence animée, approche du problème de l'implémentation

I. Partie théorique : Notions d'IFS et de séquence animée, approche du problème de l'implémentation

Introduction

Au fil de cette partie, nous allons rappeler successivement les notions d'IFS (**I**terated **F**unction **S**ystem) et de séquences animées.

Nous étudierons ce qu'il nous faut savoir sur ces IFS pour calculer leurs animations et pour pouvoir les manipuler à notre guise afin de réaliser les plus beaux effets. Nous introduirons également les principes du "théorème de collage" dans le but de permettre à tout un chacun de modéliser les fractales qu'il souhaite. Ces éléments se limiteront à ce qu'il nous faudra savoir sur les IFS pour une bonne compréhension des programmes. Cependant, les notions théoriques ne s'étendront pas vers des principes et théorèmes, certes fort intéressants et utiles, mais qui ne sont pas nécessaires dans le cadre de ce mémoire. Signalons encore que la réalisation de cette partie s'est inspirée du travail réalisé par M. Wolgang HEINEN dans son mémoire sur la génération d'images fractales. Nous pouvons en quelque sorte considérer que le travail qui suit est une continuité de son étude sur les IFS.

Ensuite, nous verrons ce qu'est une animation, quels sont les principes de base du dessin animé, du cinéma d'animation ou encore des animations de synthèse. Ceci nous permettant par la suite d'élaborer des séquences plus complexes que l'animation d'un objet d'un point initial vers un point terminal. Cette présentation de l'animation se veut introductive et ne développera pas en détail le fonctionnement d'un programme d'animation de synthèse. Ceci pourrait, par contre, être intéressant lors d'une poursuite éventuelle de l'étude des programmes élaborés dans ce mémoire afin d'améliorer les concepts utilisés et d'étendre les possibilités offertes.

Nous verrons enfin comment nous pourrons concilier ces deux théories pour arriver à réaliser des séquences animées comportant des objets fractals (dans notre cas des plantes). Ce lien entre ces théories sera déjà une première approche de la stratégie à adopter pour l'élaboration du (des) programme(s) réalisant ces animations.

Chapitre 1 : Notion d'IFS

La géométrie fractale est une extension de la géométrie classique. Elle peut être utilisée pour créer des modèles de structures physiques aussi complexes que des galaxies ou des fougères. Ainsi, il est possible de décrire la forme d'un nuage aussi précisément qu'un architecte décrirait une maison.

1.1. L'espace métrique de Hausdorff

Nous allons donc décrire un espace métrique, noté $\mathcal{H}$, et qui est l'espace où vivent les fractales. Pour cela, nous considérons que nous travaillons avec des espaces métriques (X,d) (comme par exemple ($\Re^2$, Euclidien)). Voyons dès à présent quelques définitions.

Définition 1

Soit (X,d) un espace métrique complet. $\mathcal{H}(X)$ est alors un espace où les points sont des sous-ensembles compacts et non vides de X.

Rappelons rapidement qu'un espace métrique est un couple (X,d) formé d'un ensemble X d'éléments appelés points et d'une fonction à valeurs réelles $d : X \times X \rightarrow \Re$ qui mesure la distance de tous les couples de points x et y de X et vérifie ce qui suit :

1. $d(x,y) = d(y,x) \quad \forall\, x,y \in X$
2. $0 < d(x,y) < \infty \quad \forall\, x,y \in X$ et si $x \neq y$
3. $d(x,x) = 0 \quad \forall\, x \in X$
4. $d(x,y) < d(x,z) + d(z,y) \quad \forall\, x,y,z \in X$

On appelle cette fonction ***métrique*** de l'ensemble X.

Définition 2

Soient (X,d) un espace métrique complet, $x \in X$ et $B \in \mathcal{H}(X)$. Définissons :

$$d(x,B) = \text{Min} \{ d(x,y) \text{ tq } y \in B \}$$

d(x,B) est appelé la distance du point x à l'ensemble B.

Définition 3

Soit (X,d) un espace métrique complet. Soient A, $B \in \mathcal{H}(X)$. Définissons :

$$d(A,B) = \text{Max} \{ d(x,B) \text{ tq } x \in A \}.$$

d(A,B) est appelé la *distance de l'ensemble* A à l'ensemble B.

Définition 4

Soit (X,d) un espace métrique complet. Alors la *distance de Hausdorff* entre deux points A et B de $\mathcal{H}(X)$ est définie par

$$h(A,B) = d(A,B) \vee d(B,A) = \text{Max} [d(A,B),d(B,A)].$$

Nous pouvons dès à présent considérer que tout sous-ensemble de $(\mathcal{H}(X),h)$ est une fractale. Mais présentons maintenant la complétude de l'espace fractal ainsi défini.

Théorème 1

Soit (X,d) un espace métrique complet. Alors $(\mathcal{H}(X),h)$ est un espace métrique complet. De plus, si $\{ A_n \in \mathcal{H}(X) \text{ avec } 1 \leq n < \infty \}$ est une suite de Cauchy alors

$$A = \lim_{n \to \infty} A_n \in \mathcal{H}(X)$$

avec $A = \{ x \in X \text{ tq } \exists \{ x_n \in A_n \}$ suite de Cauchy qui converge vers x $\}$.

Ce théorème nous apprend que la convergence de cette suite sera une image finale particulière puisqu'il s'agira de la fractale recherchée. Cette image finale sera donc un point de l'espace de Hausdorff que l'on appellera point ***FIXE***.

En effet, si nous considérons la figure 1, le passage d'une itération à l'autre à pour effet de recopier l'image initiale en la réduisant, en la faisant tourner et en la plaçant au centre du triangle et ainsi de suite pour les nouveaux triangles obtenus. Au bout du compte, lorsque n sera arrivé à l'infini, nous aurons dessiné l'image fractale. Cette convergence de la suite de Cauchy nous garantit ainsi l'obtention de la fractale. Cette image est réellement atteinte pour $n = \infty$, cependant on remarque en général qu'il n'est fort heureusement pas nécessaire d'attendre ce délai pour obtenir une estimation suffisante de l'image finale.

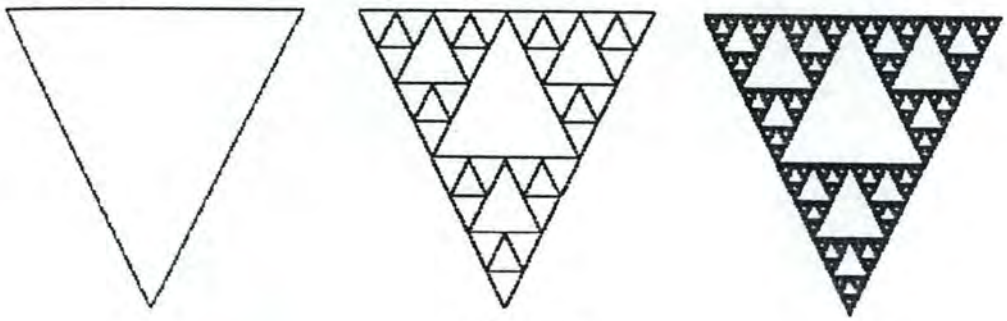

Figure 1

1.2. Les transformations affines

Les transformations affines sont introduites par les définitions suivantes.

Définition 5

Soit (X,d) un espace métrique. Une transformation de X est une application $f : X \to X$, qui à chaque point $x \in X$ associe un et un seul point $f(x) \in X$.

Définition 6

Une transformation $w : \Re^2 \to \Re^2$ de la forme :

$$w(x_1,x_2) = (ax_1 + bx_2 + e, cx_1 + dx_2 + g)$$

où a, b, c, d, e et g sont des nombres réels, est appelée *transformation affine*.

Cette expression peut s'écrire également sous la forme suivante :

$$\mathbf{W(x) = Ax + t}$$

avec $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ et $t = \begin{pmatrix} e \\ g \end{pmatrix}$.

Si nous analysons cette formule, nous pouvons en tirer qu'une transformation affine se compose d'une transformation linéaire définie par la matrice A suivie par une translation représentée par t.

Nous allons maintenant pouvoir présenter la notion suivante qui est la notion de transformation contractive dans l'espace de Hausdorff. Voyons donc par l'intermédiaire des deux lemmes qui suivent ce que nous entendons par là.

Lemme 1 :

Soit $w : X \to X$ une transformation contractive d'un espace métrique (X,d) avec un facteur de contractivité s. Alors $w : \mathcal{H}(X) \to \mathcal{H}(X)$ définie par :

$$\mathbf{w(B)} = \{ w(x) : x \in B \} \quad \forall B \in \mathcal{H}(X)$$

est une transformation contractive de $(\mathcal{H}(X),h(d))$ avec facteur de contractivité s.

Rappelons ici qu'une transformation contractive $f : X \to X$ d'un espace métrique (X,d) est telle qu'il existe une constante s, appelé facteur de contractivité, avec $0 \leq s \leq 1$ et impliquant : $\mathbf{d(f(x),f(y)) \leq s\ d(x,y)\ ,\forall\ x,y \in X.}$

Lemme 2 :

Soient (X,d) un espace métrique et $\{ w_n : n = 1,2,...N \}$ un ensemble de transformations contractives de $(\mathcal{H}(X),h(d))$ de facteurs de contractivité s_i respectifs. Si $W : \mathcal{H}(X) \to \mathcal{H}(X)$ est définie par :

$$\mathbf{W(B) = w_1\ (B)\ U\ w_2\ (B)\ U\ ...\ U\ w_N\ (B)} \quad \forall B \in \mathcal{H}(X)$$

alors W est une transformation contractive avec facteur de contractivité $s = \text{Max} \{ s_i : i = 1,2...N \}$.

A partir de ces lemmes, nous allons pouvoir définir les deux nouvelles notions que sont ***IFS*** et ***ATTRACTEUR.***

Définition 7

Un IFS consiste en un espace métrique (X,d) et un ensemble fini de transformations contractives $w_n : X \to X$ avec comme facteurs de contractivité respectifs s_n pour n = 1,2...,N. IFS signifie *Iterated Function System*. Sa notation est {X; w_n pour n = 1,2,...,N } et son facteur de contractivité est s = Max {s_n pour n = 1,2,...,N }.

Voyons maintenant un théorème qui englobera l'ensemble des notions citées jusqu'à présent et qui définit ce qu'est un attracteur. Il s'agit en fait de la notion de point fixe que nous avons développée précédemment dans le cadre des transformations contractives définies ci-dessus.

Théorème 2

Soit { X, wn, n = 1,2...,N } un IFS avec facteur de contractivité s. Alors la transformation $W : \mathcal{H}(X) \to \mathcal{H}(X)$ définie par :

$$W(B) = \cup w_n(B) \quad (n=1,...,N)$$

est une transformation contractive de l'espace métrique complet $(\mathcal{H}(X),h(d))$ avec facteur de contractivité s. Nous avons donc :

$$h(W(B),W(C)) \leq s\, h(B,C)$$

pour tous $B,C \in \mathcal{H}(X)$. Le point fixe sera $A \in \mathcal{H}(X)$ et vérifiant :

$$A = W(A) = \cup w_n(A) \quad (n=1,...,N)$$

Il est unique et obtenu par $A = \lim_{n \to \infty} W^n(B)$ $\forall B \in \mathcal{H}(X)$.

Par définition, nous appellerons le point fixe unique A ainsi obtenu l'attracteur de l'IFS.

1.3. Les algorithmes

Ils sont au nombre de deux et se nomment l'algorithme déterministe et l'algorithme du chaos (Random Iteration Algorithm). Le résultat obtenu par ces deux algorithmes est identique. Il s'agit bien évidemment de l'attracteur de l'IFS qui représente en fait l'image fractale souhaitée. Cependant leurs principes et leur utilisation diffèrent quelque peu.

1.3.1. L'algorithme déterministe

Le principe de cet algorithme est qu'il part d'un ensemble initial A_1 de $\Re 2$ et qu'il va au fur et à mesure calculer l'ensemble suivant résultant de l'application de la transformation à l'ensemble initial. Ceci s'exprime en considérant $\{X; w_n, n=1,...,N\}$ un IFS et en calculant A_{n+1} à partir de A_n comme suit :

$$\mathbf{A_{n+1} = U\ w_i\ (A_n) = W\ (A_n)} \quad (i = 1,..,N \text{ et } n = 1,2...)$$

La convergence garantissant l'obtention d'un attracteur, le résultat sera obtenu pour $n = \infty$ et sera l'image fractale.

Voici l'algorithme déterministe en détail :

Algorithme 1

```
initialiser bitmap1(x,y) et bitmap2(x,y)
initialiser transformations affines
                    a,b,c,d,e,g(nt)
                    nt = nbre de transformations
initialiser A1 (ensemble de départ) dans bitmap1

pour n = 1 à nbre_itérations
      pour i = 1 à x faire
      pour j = 1 à y faire
         appliquer W à An ⇒ An+1 c-à-d
           si bitmap1(i,j) = 1 alors
            bitmap2(a(1)*i+b(1)*j+e(1), c(1)*i+d(1)*j+g(1)) = 1
```

```
                    bitmap2(a(2)*i+b(2)*j+e(2), c(2)*i+d(2)*j+g(2)) = 1
                    ...
                    bitmap2(a(nt)*i+b(nt)*j+e(nt), c(nt)*i+d(nt)*j+g(nt))=1
                fin si
            fin j
            fin i

            dessiner bitmap2
            recopier bitmap2 dans bitmap1

        fin n
```

Si nous analysons un peu cet algorithme, nous constatons qu'après initialisation des paramètres de départ, il dessine l'image initiale dans la bitmap 1. Ensuite les itérations vont commencer et à chacune d'entre elles, le dessin de l'image suivante va se réaliser dans la bitmap 2 pour être ensuite affiché et recopié dans la bitmap 1. Le processus pourra alors recommencer jusqu'à atteindre un nombre d'itérations suffisant procurant un dessin, disons plutôt une approximation de l'attracteur final. En effet, l'attracteur ne serait obtenu qu'après une infinité d'itérations.

Le schéma ci-dessous (Figure 2) reprend quelques étapes de cet algorithme dans le cas d'une fougère, l'image initiale étant ici un rectangle. Le second dessin donne le résultat à l'itération 1, le troisième à l'itération 5 et le dernier après 10 itérations.

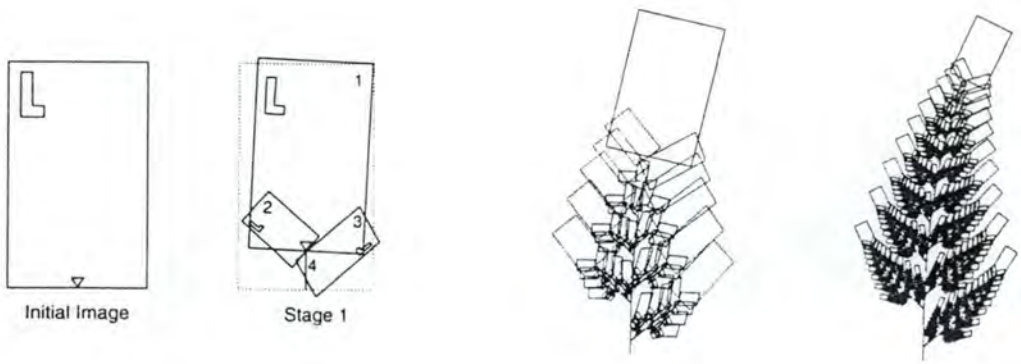

Figure 2

Ce qui ressort immédiatement de ces schémas est qu'il faut tout d'abord un nombre d'itérations relativement élevé avant d'obtenir une bonne approximation du résultat final. Nous entendons par bonne approximation une image représentative de la fractale souhaitée (dans notre cas, une fougère). De plus, l'algorithme est conçu par un travail sur deux bitmaps. Ceci impose au programmeur un certain gaspillage de place mémoire pour la sauvegarde permanente de ces données.

Nous pourrions donc conclure que cet algorithme, certes efficace, n'est pas le plus adéquat pour la réalisation de nos animations. En effet, le temps de calcul étant assez élevé pour l'obtention d'une image, il pourrait ralentir fortement le calcul de toute une animation.

1.3.2. L'algorithme chaos

Cet algorithme s'apparente fort au précédent mais cette fois nous allons associer à chaque w_i une probabilité P_i. Nous avons maintenant un IFS défini par { X; w_n , n=1,..,N } et { P_i , i=1,..,N} avec les probabilités qui vérifient :

$$\mathbf{P_1 + P_2 + ... + P_N = 1 \ \ et \ \ P_i > 0 \ \ (pour \ i = 1,2...)}$$

Ces probabilités jouent un rôle important dans le calcul des images de l'attracteur d'un IFS. Cette fois, l'algorithme qui suit va dessiner directement l'attracteur en faisant courir un point sur celui-ci. Cependant ce point ne passe pas identiquement partout sur l'attracteur mais plus particulièrement à certains endroits en fonction des probabilités choisies.

Voici l'algorithme :

```
initialiser :
        a,b,c,d,e,g(nt)
        nt = nbre de transformations
        p(nt) (les probabilités)

définir un point initial (x,y)
```

```
pour n = 1 à nbre_itérations
        choisir une transformation au hasard = t
        déterminer le nouveau point :
                newx = a(t)*x + b(t)*y + c(t)
                newy = d(t)*x + e(t)*y + g(t)
        x = newx et y = newy
        dessiner le point (x,y)
fin n
```

Signalos que la démonstration de la convergence de cet algorithme s'obtient en considérant que l'on construit en fait une suite de Cauchy convergeant vers l'attracteur de l'IFS [1].

La différence essentielle avec l'algorithme précédent est qu'il ne nécessite pas la réservation de grandes quantités de mémoire pour le calcul de l'image. En effet, il suffit ici de conserver comme données la valeur d'un point. De plus, le calcul se réalise beaucoup plus rapidement, l'obtention de l'attracteur est obtenu après un nombre d'itérations bien inférieur à l'infini. Ces deux avantages nous feront opter ultérieurement pour ce dernier algorithme afin d'optimiser le calcul des animations. La figure 3 nous montre trois étapes d'avancement du calcul de l'IFS.

Figure 3

[1] La démonstration de la convergence de cette suite est présentée dans le mémoire "Génération d'images fractales et implémentation en langage objet" de M. Heinen (cf. Bibliographie).

Mais attardons-nous quelque peu sur l'avantage de ces probabilités. Considérons la figure ci-dessous. Il s'agit du dessin d'une fougère réalisé avec l'algorithme-chaos avec 100000 points. L'image de gauche est obtenue avec des probabilités égales pour toutes les transformations, et l'image de droite avec des probabilités différentes. Ce choix approprié des probabilités impose au point de parcourir plus souvent certaines transformations et moins d'autres. Sur l'image de gauche, le point a tendance à rester le long de la tige, tandis qu'à droite les probabilités le forcent à parcourir les extrémités des feuilles.

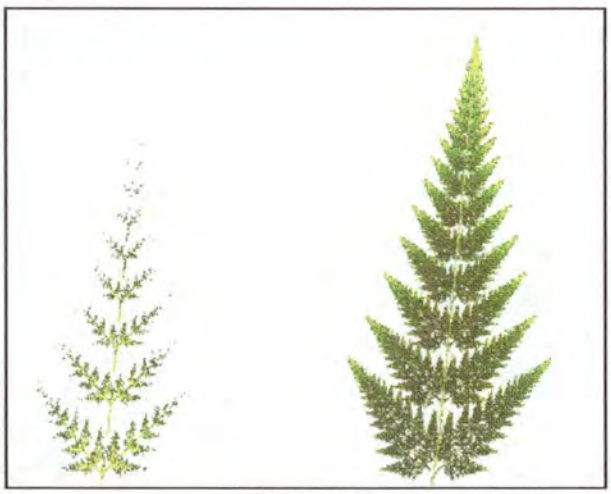

Figure 4

1.4. Dépendance continue des IFS par rapport à leurs paramètres

Cette dépendance s'exprime au sein du théorème suivant :

Théorème 3

Soit (X,d) un espace métrique. Soit { X; (w_0), w_1, w_2, ..., w_N } un IFS avec s comme facteur de contractivité. Pour n = 1,...,N soit wn dépendant continûment du paramètre p ∈ P (où P est un espace métrique compact). Alors l'attracteur A(p) ∈ H(X) dépend continûment de p ∈ P et ceci en respectant la métrique de Hausdorff h(d).

En d'autres termes, ce théorème exprime que de petites modifications apportées aux paramètres de départ de l'IFS engendrent de petits changements à

l'image finale. Ceci est très important car en effet, cela nous permet de conserver continuellement un contrôle sur l'attracteur en ajustant les paramètres de la transformation. Il s'agit de la notion la plus importante en ce qui nous concerne puisqu'elle nous apprend qu'il est possible d'obtenir une image légèrement différente sur certains points simplement en modifiant quelques paramètres de la transformation.

En plus de cela, nous allons voir dans les pages qui suivent comment trouver ces paramètres de la transformation à partir d'une image ou d'un dessin. Pour ce faire, nous nous baserons sur le théorème dit de collage. La particularité de cette continuité dans la dépendance entre les paramètres et l'attracteur nous permet déjà de penser que le calcul de ces paramètres de départ ne devra pas nécessairement atteindre une grande précision pour que le résultat final s'approche de l'image souhaitée.

La figure 5 qui suit nous présente diverses planches où sont repris les attracteurs d'un IFS de type fougère en trois dimensions pour lequel diverses modifications des paramètres ont été opérées. Le résultat est très satisfaisant et nous permet déjà d'envisager le type de stratégie qu'il faudra choisir afin de pouvoir réaliser une animation.

Figure 5

1.5. Le théorème de collage

Jusqu'à présent, il nous était possible sur base d'un ensemble de transformations de créer une image fractale, l'attracteur de l'IFS défini par les transformations. Mais nous pourrions nous demander comment obtenir ces mêmes transformations sur base d'un dessin ou d'une image. Pour obtenir ce mécanisme inverse, nous allons étudier le théorème suivant appelé théorème de collage :

Théorème 4 (de collage)

Soit (X,d) un espace métrique complet. Soit $L \in \mathcal{H}(X)$ donné et soit $\varepsilon > 0$. On choisit un IFS $\{ X; w_n, n = 1,...,N \}$ de facteur de contractivité s $(0 \leq s \leq 1)$ tel que :

$$h\left(L, \bigcup_{i=1}^{n} w_i(L)\right) < \varepsilon \quad , \text{ où } h(d) \text{ est la distance de Hausdorff.}$$

Alors :

$$h(L, A) \leq \varepsilon / (1 - s) \quad , \text{ où A est l'attracteur de l'IFS}$$

D'un point de vue pratique, l'application de ce théorème pour trouver les transformations contractives qui définissent l'IFS pourra paraître quelque peu artisanal et pourtant...

Il suffit de faire recouvrir notre image par un ensemble de plusieurs copies similaires mais réduites. A la figure 6, par exemple, la fougère a été recouverte par quatre autres fougères identiques en réduction. C'est ainsi que l'on définit un IFS formé de quatre transformations. Chaque transformation est définie par six scalaires (a,b,c,d,e,g). Pour trouver ces six nombres, prenez quatre points de la fougère, puis leurs images dans la fougère réduite et procédez à la résolution du système d'équations ainsi obtenu. Vous recommencez la même opération pour chacune des transformations et vous obtiendrez les valeurs recherchées. Quant aux probabilités, elles seront, elles aussi, choisies de façon approximative. Le plus simple est de choisir le rapport de l'aire de l'image de la figure par la transformation w_i et de l'aire de la figure elle-même :

$$\text{Aire } (w_i \text{ (Figure)}) \text{ / Aire (Figure).}$$

Il faudra s'assurer qu'aucune probabilité n'est nulle. Si cela était le cas, la transformation correspondante ne servirait à rien. De plus, la somme de ces probabilités doit valoir 1.

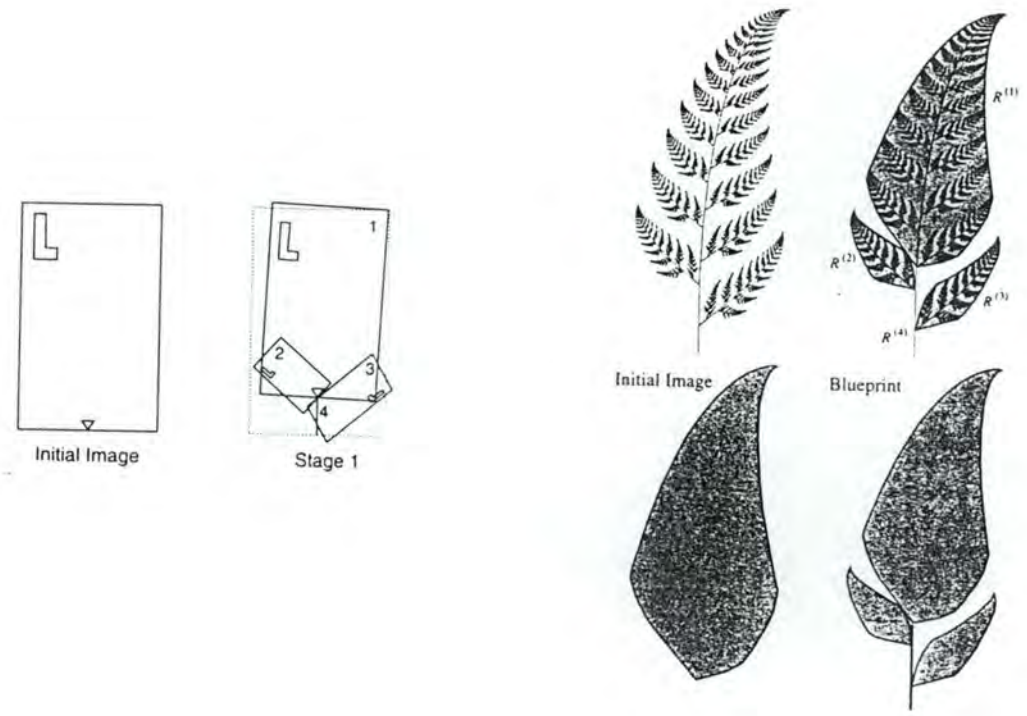

Figure 6

Cette technique pour trouver les valeurs des transformations est suffisamment efficace pour obtenir de bons résultats. En effet, comme nous l'avons déjà vu, la dépendance continue de l'IFS par rapport à ces paramètres nous a appris que de petites variations de ces paramètres n'engendraient que de petits changements sur l'attracteur. Ainsi, même si les paramètres sont légèrement différents, ils représentent malgré tout une bonne approximation de l'image souhaitée.

Chapitre 2 : Notions de séquence animée...

L'animation peut être définie de différentes façons. Pour John Halas, un célèbre concepteur d'animations, "le mouvement est l'essence de l'animation". Une approche similaire définit l'animation comme "l'art du mouvement". D'autres définitions plus complètes sont données ci-dessous :

- L'animation est une technique dans laquelle l'illusion de mouvement est créée par la photographie d'une série d'images individuelles sur un film en une succession de plans. L'illusion est obtenue en projetant le film à une certaine vitesse (généralement 24 images/seconde).

- L'animation revient à un processus dynamique générant une série d'images d'un ensemble d'objets, où chaque image est une légère modification de l'image précédente.

Toutes ces définitions décrivent le principe d'animation comme il a été conçu il y a 80 ans, elles sont toujours valides aujourd'hui dans la plupart des cas. Généralement, l'animation est basée sur la technique dit de "*l'image par image*". Les animations sur ordinateur sont souvent réalisées en utilisant la même stratégie. Cependant, dans le cas précis de l'animation en temps réel, les définitions (et tout particulièrement la première) sont incomplètes. Par exemple, les jeux vidéos sont relativement différents des produits plus conventionnels issus de l'imagerie informatique. De plus, cette première définition assimilant l'animation au mouvement n'est pas complète. En effet, l'animation peut exister sans mouvement, par exemple :

- dans une métamorphose où un objet se transforme en un autre
- dans un changement de couleur (le héros devient rouge de honte)
- dans un changement de l'intensité lumineuse (un coucher de soleil).

Habituellement, l'animation est orientée vers la production de dessins en deux dimensions. Chaque *'frame"* est une image plate et est dessinée à la main. Ces

dessins sont complexes et nécessitent souvent des équipes complètes pour la réalisation.

Décomposition d'un film :

Un dessin animé peut se décomposer en divers éléments qui sont : le résumé en quelques pages, le scénario complet, le storyboard, les séquences composées de scènes elles-mêmes composées d'images. Il est important de remarquer donc qu'un film animé est composé de séquences qui définissent des actions spécifiques, chaque séquence consistant en une série de scènes qui sont généralement définies par un ensemble de personnages ou un paysage particulier... Les scènes sont alors divisées en images. Cette décomposition est reprise dans la figure 7 ci-dessous.

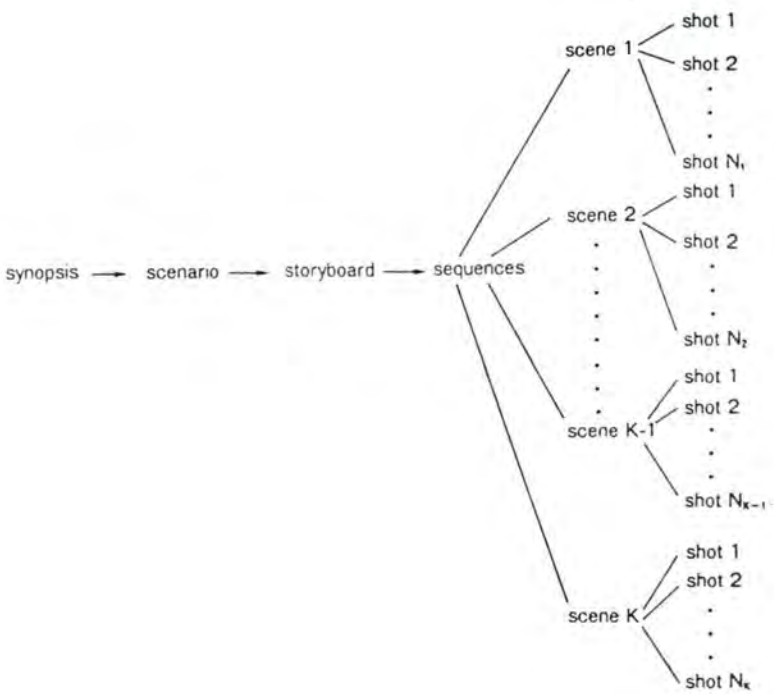

Figure 7

Parmi les autres notions que nous pourrions retenir de la réalisation d'un dessin animé, il y a la bande son, élément indispensable à l'ambiance rendue par l'animation. Il faut également souligner l'importance de la coloration des séquences, réalisme supplémentaire assuré. Mais le plus important consiste en la définition de "***positions clés***". Il s'agit de points intermédiaires dans l'évolution de la séquence.

Ces points définissent les grands changements s'opérant sur un personnage, un paysage, une palette de couleur... A partir de là, on définit ce que l'on pourrait appeler les "positions-entre" qui représentent l'ensemble des étapes intercalées entre deux points intermédiaires et assurant le passage fluide du premier au second point. Cette notion est primordiale dans l'approche de notre problème d'animation fractale. C'est en effet sur elle essentiellement que nous allons baser notre stratégie de création de mouvement. Pour comprendre ce que nous entendons par points intermédiaires, il suffit de regarder les planches ci-dessous définissant les positions clés du mouvement d'une main se repliant. A partir de là, c'est l'ordinateur qui calculera les 10, 15 ou 50 images situées entre chaque position clé.

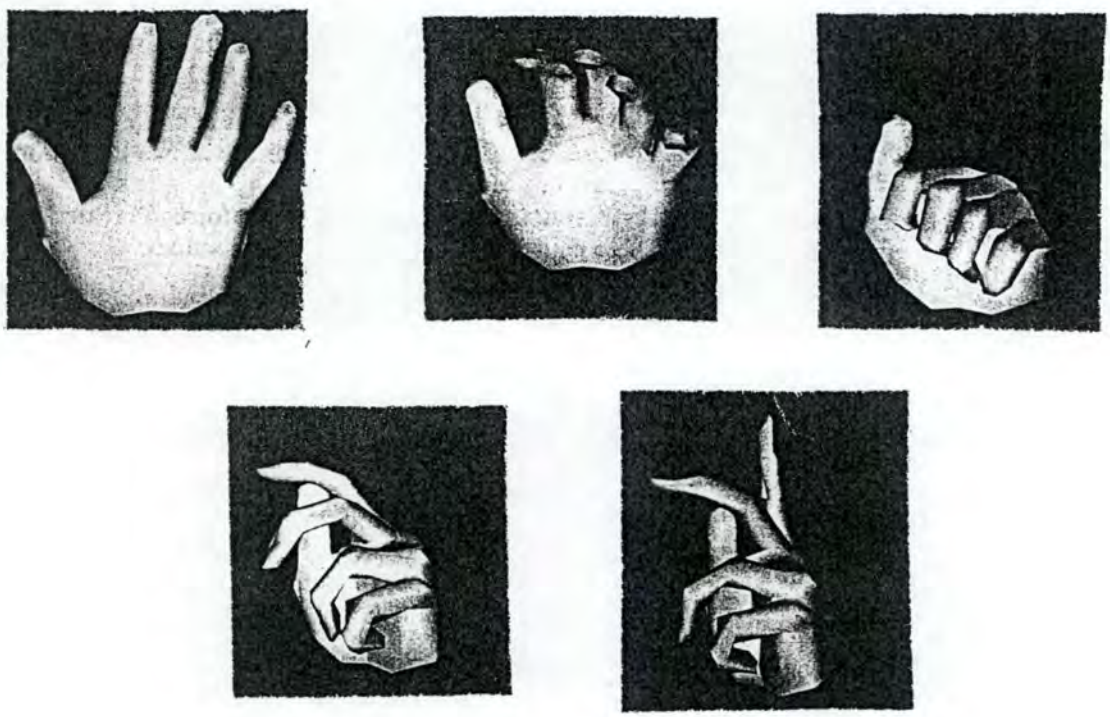

Figure 8

Applications pour les animations :

Voici quelques utilisations courantes d'animations qui peuvent être rassemblées en cinq catégories :

- **La télévision** : utilise les animations pour des titres, des logos ainsi que des dessins animés...

- **Le cinéma** : Les animations y ont souvent joué un rôle important. Elles permettent l'obtention d'effets spéciaux surprenants.

- **Les gouvernements** : L'utilisation qui est faite des animations est réalisée à des fins publicitaires pour la communication aux masses.

- **L'éducation et la recherche** : peuvent utiliser intensivement les animations à des fins éducatives. L'animation peut être d'une grande aide dans la réalisation de simulateurs en tout genre (médecine, armée...)

- **L'industrie** : Le rôle de l'animation y est fort semblable à celui du gouvernement (publicité, relation publique...)

L'animation sur ordinateur :

L'ordinateur peut jouer une variété impressionnante de rôles différents dans la réalisation d'animations. Il peut ainsi être un assistant précieux dans la représentation d'objets à l'aide du dessin (classique ou tridimensionnel). Il peut aider à la colorisation de scènes ou encore à la synchronisation lors d'un montage. Mais il peut surtout calculer complètement les images intermédiaires situées entre deux positions clés et ainsi des animations complexes peuvent être obtenues par calcul automatique de la majorité des images. Il reste donc à définir à la machine ces fameuses positions clés, que nous appellerons également points intermédiaires.

"Temps réel" contre "Image par image" :

Peut-on dire qu'un ordinateur est utilisé dans la réalisation d'une animation simplement pour produire chaque image individuellement pour être ensuite "photographiée" ? Ou le film est-il produit directement à l'écran ? Tout cela est une question de *temps*.

Pour comprendre cela, nous allons effectuer la comparaison entre les séquences calculées par ordinateur du film 'TRON' et un jeu vidéo comme 'PacMan' ou 'Super Mario'. Les deux peuvent être considérés comme étant des animations. Cependant, dans le film 'TRON' les images sont très complexes et réalistes. Cela

signifie une nécessité d'attendre plusieurs minutes pour produire une image. Ces images doivent alors être enregistrées (photographiées) et ensuite projetées à une vitesse de 24 images/seconde. D'autre part, dans un jeu vidéo l'animation est immédiate. Les objets bougent rapidement et il existe une interaction complète entre l'utilisateur et l'image. Il s'agit d'animation en "temps réel" parce que le moment séparant la décision de mouvement de l'utilisateur et la réalisation du mouvement est un laps de temps très court. L'animation en temps réel ne nécessite pas d'enregistrement sur film puisque son résultat s'obtient instantanément sur l'écran.

Cette animation en temps réel est limitée par les performances des ordinateurs. En effet, les images doivent être affichées en moins de 1/10 de seconde parce que l'illusion de continuité du mouvement est rompue pour des vitesses inférieures. Il s'agit d'une limitation exigeante car uniquement de simples calculs peuvent être réalisés dans des délais aussi brefs. La contrainte est imposée par le cycle horloge de l'ordinateur, les capacités de stockage, la longueur des instructions...

Un signe cependant des nouvelles possibilités pour la réalisation d'animation temps réel est constituée par les nouvelles avancées technologiques réalisées dans le développement de matériel adapté, comme le multi-processing en tableau ou encore les processeurs graphiques.

Le Playback en temps-réel et le buffer-image :

L'illusion d'animation en temps réel peut être créée avec un ***buffer-image*** qui peut s'apparenter à un tableau bi-dimensionnel de pixels. Ce buffer se remplit des images calculées au fur et à mesure de l'avancement dans l'animation. L'alternative est alors d'utiliser la technique du ***playback*** qui consiste à afficher ces images conservées dans le buffer-image. Cet affichage étant rapide, l'illusion de mouvement recherchée est ainsi obtenue. Notons que ce buffer-image est défini dans la mémoire de masse qui peut être un disque ou la mémoire centrale...

Chapitre 3 : Précisons notre choix de stratégie

Il est certain que l'ensemble de la théorie que nous avons présentée ne pourra être appliquée dans le cadre de ce mémoire et au sein d'un seul programme. Il va donc nous falloir choisir les éléments les plus importants et les plus utiles pour notre réalisation d'animation.

Nous n'allons donc retenir de ce qui précède qu'un certain nombre de notions. Notre objectif étant la réalisation d'une application offrant des animations fractales, il nous suffit de considérer les éléments de base de la séquence animée ainsi que la théorie sur les IFS.

Nous baserons donc notre stratégie sur l'algorithme chaos permettant la modélisation de l'attracteur d'un IFS prédéfini. Nous utiliserons alors la notion de séquence animée contenant un certain nombre de scènes formées d'images. Ces scènes correspondront à des positions clés représentant un changement fondamental dans la forme de l'attracteur et donc de l'image. A partir de là, le programme devra être à même de calculer l'ensemble des images permettant le passage d'un point intermédiaire à un autre, le tout donnant l'animation finale.

Pour apporter un maximum de réalisme aux animations, nous allons choisir également de générer des images colorées sur une palette représentative de couleurs. Pour ce faire, nous nous baserons sur les définitions des IFS desquels nous pourrons faire ressortir les propriétés nécessaires à la coloration d'un attracteur.

Notons enfin que la plupart du temps, il est conseillé d'afficher au moins 24 images/seconde pour obtenir une bonne animation. mais comme nous l'avons vu, nous pouvons considérer qu'un taux de 10 images/secondes peut être suffisant. C'est sur ce taux que nous baserons nos calculs et nos animations ultérieures.

Il est certain que nous pourrions développer énormément de possibilités pour obtenir des animations encore plus complexes et plus réalistes. Cependant, nous nous contenterons ici de jeter les bases nécessaires à la réalisation de notre programme. Dans le cadre d'autre travail, il serait alors possible d'approfondir le sujet et d'étoffer le noyau central que représente notre application. Il serait par exemple intéressant de gérer plusieurs objets simultanément, de prévoir des fonds d'écran, d'ajouter éventuellement du son...

Conclusion

Nous voilà donc arrivé au terme de cette première partie consacrée à la théorie. Nous avons maintenant les notions nécessaires à la réalisation de notre projet ainsi qu'une idée assez précise de notre stratégie. Cette présentation théorique des objets manipulés ne se voulait absolument pas exhaustive. Il existe encore bien de longues théories sur le sujet et de nombreux théorèmes pouvant apporter un "plus" aux notions déjà présentées. Cependant, le temps qui nous était imparti ne nous a malheureusement pas permis d'approfondir ces recherches plus loin.

Nous allons dès à présent passer à notre seconde grande partie. Il s'agit de la partie pratique au cours de laquelle nous allons présenter le développement des programmes que nous avons mis au point. Nous verrons alors comment réellement utiliser les notions que nous avons étudiées jusqu'à présent.

PARTIE PRATIQUE

Diverses implémentations d'animations fractales

II. Partie pratique : Diverses implémentations d'animations fractales

Introduction

Après avoir, dans une première partie, tenté de cerner le problème central de l'animation d'images fractales de type IFS, nous passons maintenant à la partie pratique au cours de laquelle nous allons aborder le problème de l'implémentation.

Notre étude va se porter dans un premier temps sur une implémentation en Turbo Pascal sous Dos. Cependant, les limitations imposées par la programmation sous Dos en ce qui concerne la gestion d'images de type bitmap vont nous conduire à choisir l'environnement Windows pour réaliser l'application finale.

Nous allons donc étudier les principes de bases des animations sur quelques programmes Dos et ensuite appliquer les mêmes algorithmes à la programmation sous Windows mais en utilisant, cette fois, la puissance offerte pour la manipulation de bitmaps et la gestion de la mémoire. Cette application finale sera ensuite exposée et décortiquée afin d'en comprendre son utilisation. Et pour terminer, nous donnerons quelques exemples d'animations calculées avec le programme. Cependant, comme vous vous en doutez, ces animations resteront fixe...

On peut déjà à ce stade signaler que l'exécution de ces quelques programmes peut nécessiter des machines "musclées" au niveau de la mémoire, du processeur central et des cartes graphiques. Mais nous reviendrons à ces problèmes ultérieurement.

Signalons enfin que les listings des programmes correspondant à notre analyse sont repris en annexe en fin de mémoire et qu'en plus, vous trouverez au bout de cette annexe une disquette contenant ces mêmes programmes.

C'est après que nous conclurons notre partie pratique.

Chapitre 1 : Conception des programmes et algorithmes de base

L'algorithme-chaos de génération d'images pour un IFS défini par ses transformations contractives et ses probabilités a déjà été exposé dans la première partie. Nous allons l'appliquer dans un premier temps aux paramètres d'une fougère en deux dimensions et ceci pour une animation simple formée uniquement d'un point initial et d'un point terminal.

Programme n° 1

Nous partons donc du point Pt_1 et nous allons calculer un certain nombre d'images intermédiaires avant d'arriver au point Pt_2. Pour ce faire, il faut bien évidemment définir l'ensemble des paramètres et probabilités de l'image en Pt_1 et en Pt_2. Une fois ces données définies, il suffira de calculer la différence pour chaque paramètre entre le Pt_2 et Pt_1 et de la diviser par le nombre d'images à calculer pour l'animation. Nous appellerons ces valeurs 'écart moyen'. Il restera alors, à chaque calcul d'image, à les ajouter aux paramètres de l'image précédente pour obtenir la nouvelle image. Au bout du compte, les images ainsi dessinées au fur et à mesure donneront l'impression d'animation de l'objet.

Voyons donc ce qu'est devenu l'algorithme de calcul de l'attracteur d'un IFS. Les modifications sont notées en italiques.

Algorithme 3

Initialiser les transformations des points Pt_1 *et* Pt_2 :
a1,b1,c1,d1,e1,f1,*a2,b2,c2,d2,e2,f2[nt]*
nt = nbre de transformations
les probas associées = p1,*p2[nt]*
NbImage = nbre d'images souhaitées pour l'animation

Calcul de l'écart moyen entre les valeurs extrêmes :
aint,bint,cint,dint,eint,fint,pint où :
aint[i] = (a2[i] - a1[i]) / NbImage
...
fint[i] = (f2[i] - f1[i]) / NbImage
pint[i] = (p2[i] - p1[i]) / NbImage

Choisir un point initial (x,y)

Pour Nbim = 1 à NbImage
Effectuer l'ajout de l'écart moyen à l'image précédente :
Pour i = 1 à nt
a[i] = a[i] + aint[i]
b[i] = b[i] + bint[i]
...
p[i] = p[i] + pint[i]

Pour n = 1 à nbre_point
choisir une transformation au hasard = t
(en fonction des probas)
déterminer le nouveau point :
newx = a[t]*x + b[t]*y + e[t]
newy = c[t]*x + b[t]*y + f[t]
x = newx, y = newy
dessiner le point (x,y)
Fin n
Fin Nbim.

Cet algorithme est utilisé pour le programme 'Farn2D' se trouvant en annexe et réalisé en Turbo Pascal. La particularité est qu'il demande d'abord à l'utilisateur le nombre d'images qu'il souhaite pour l'animation ainsi que le nombre de points à calculer pour chaque image. Le type d'animation choisi est la croissance d'une fougère en deux dimensions. Cette fougère part donc de rien pour arriver à sa forme finale.

Lorsque l'on utilise ce programme, on se rend bien vite compte qu'il met *'énormément'* de temps pour le calcul de chaque image. Ce délai est évidemment fonction du type de processeur utilisé pour effectuer le calcul. Cependant même sur un 486 DX-50 Mhz, le programme traîne... C'est pour cette raison que l'on demande au départ le nombre de points à calculer par image et ceci afin d'obtenir malgré tout une petite idée de l'animation avec une certaine fluidité.

C'est donc ici que l'on fait ressortir le premier problème rencontré pour la réalisation d'animation (et le plus important). Il s'agit du temps nécessaire au calcul d'une image contenant beaucoup de points. L'obtention d'animation en temps réel sur nos modestes PC n'est pas encore pour aujourd'hui, ni même pour demain. Cependant, signalons que l'algorithme-chaos de calcul d'IFS se prête particulièrement bien au multi-processing et on peut alors escompter arriver à du calcul en temps réel sur des machines dont l'architecture de base se compose de plusieurs processeurs.

Pour pouvoir contourner ce problème, il suffirait de calculer toutes les images au préalable, de les stocker et ensuite de les afficher rapidement les unes après les autres. Cela nous apporterait la fluidité recherchée mais nous pose un nouveau problème, celui du stockage des informations.

La première idée est alors de stocker ces images dans des tableaux définis en mémoire centrale, chaque case du tableau représentant un pixel à l'écran (cf. Figure 9). Cependant, le nombre de tableaux de ce type doit se limiter à quatre ou cinq tableaux de 100x100 ce qui nous donne une animation de très courte durée et sur une petite portion de l'écran. Cette idée est donc à rejeter.

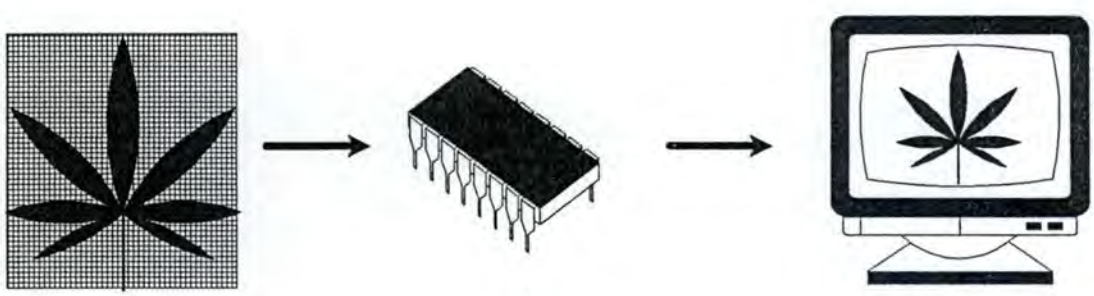

Figure 9

La seconde tentative fut de stocker chaque image calculée sur disque dur au format bitmap et ensuite d'aller rechercher ces images pour les afficher *rapidement* à l'écran (cf. Figure 10). Une fois de plus, le résultat fut décevant. Les images étaient certes plus grandes (environ 300 x 350 pixels), mais cette fois la place prise sur disque dur par une animation d'une trentaine d'images était considérable (de l'ordre de **3,5** Mégabytes). De par ce fait, lorsqu'il fallait recharger ces images à partir du disque dur pour les afficher à l'écran, nous retrouvions une saccade due cette fois à la **'lenteur' du périphérique** qui arrivait à charger le tout en plus ou moins **5-6** secondes pour une animation qui devrait durer tout au plus 3 secondes. Il y avait une certaine amélioration mais elle n'était pas encore suffisante. Cette solution pourrait être suffisante lorsque l'on utilise un disque d'une certaine capacité avec un taux de transfert supérieur à 1200-1300 Kilobytes/seconde. Remarquons ici que ce taux doit être le taux réel et non pas le taux apparent fourni par certains tests de hardware qui obtiennent des résultats biaisés de par le fait que le disque dur contient un petit peu de mémoire cache. Cette mémoire, de l'ordre de 32 ou 64 Kilobytes, n'a évidemment aucune utilité dans le cadre du transfert de nos images.

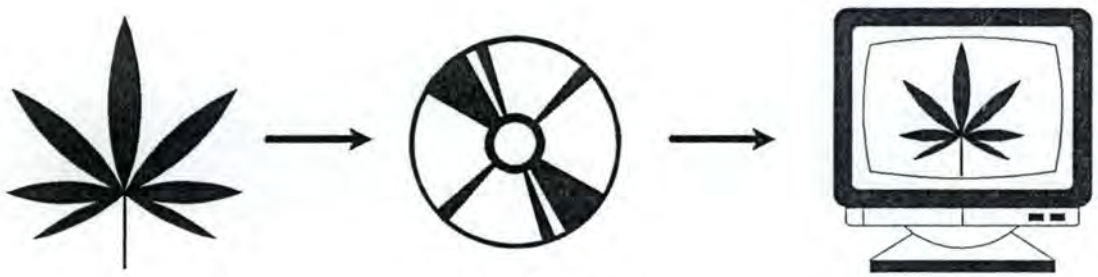

Figure 10

Enfin, la dernière idée d'essai fut de stocker ces images en mémoire (cf. Figure 11). Avec une mémoire suffisante cela aurait permis d'afficher les images beaucoup plus rapidement qu'en les mettant sur disque dur. Mais cette fois, la limitation est venue du langage en lui-même. Le Turbo Pascal sous Dos, de même que d'autres langages de programmation sous Dos, permet la sauvegarde de **deux** images bitmap à la fois en mémoire centrale. Ceci ne permet évidemment pas la réalisation d'animation d'une trentaine d'images et donc les essais ne furent même pas réalisés.

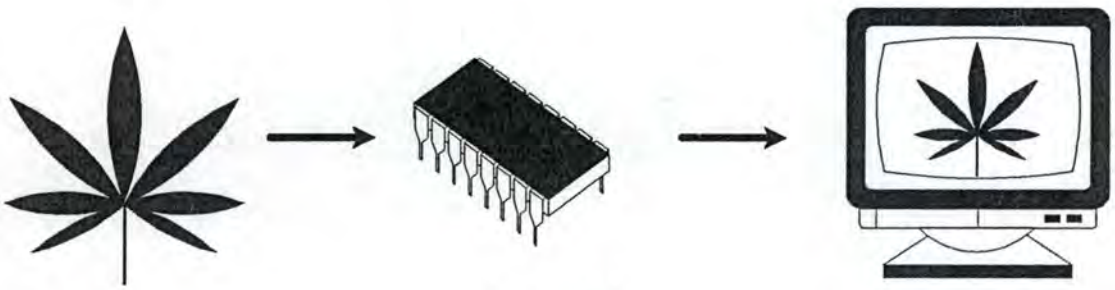

Figure 11

Une dernière remarque : lorsqu'on travaille sous Dos, il est possible d'afficher des images dans des résolutions de 320 x 200 pixels par exemple ou de 640 x 480 pixels. La première résolution étant trop basse, le choix pour ces programmes s'est porté sur le **640 x 480** pixels. Cependant, cette résolution se limite à un affichage simultané de **16 couleurs** à l'écran. Ceci représente une limitation supplémentaire qu'il faudrait dépasser.

Nous pouvons donc conclure, à ce stade, que la réalisation d'un tel programme nécessite des machines à **configuration musclée**, que ce soit au niveau processeur ou aux niveaux mémoire et disque dur. Nous verrons par la suite quelques tests qui ont été effectués sur certains types de PC.

Programme n° 2

Nous allons cette fois appliquer la notion de point intermédiaire dans une séquence animée, notion que nous avons développée dans la partie théorique. Pour ce faire, il nous faut considérer non plus un point initial et final, mais un point initial Pt_1, un point final Pt_n et une succession de points intermédiaires Pt_i ($1<i<n$). Notons que les points Pt_1 et Pt_n sont également appelés points intermédiaires. Comme précédemment, il faut définir les paramètres des IFS de chaque point intermédiaire, ainsi que les probabilités respectives. A partir de là, il faut calculer l'écart moyen qui est cette fois défini entre chaque point intermédiaire. Cet écart est obtenu comme auparavant en calculant la différence entre les valeurs de Pt_i et Pt_{i-1} et en divisant par le nombre d'images souhaité. L'ajout à chaque itération de cet

écart aux paramètres de l'image précédente permet à nouveau de calculer l'image suivante et d'obtenir l'animation.

Cette fois l'algorithme 3 se transforme comme suit (transformations en italique) :

Algorithme 4

Initialiser les transformations des points Pt_i *(1 ≤ i ≤ NbPointInter)* :
a,b,c,d,e,f[i,nt]
nt = nbre de transformations
les probas associées = *p[i,nt]*
NbImage = nbre d'images souhaitées
NbPointInter = nbre de points intermédiaires

Pour boucle = 1 à (NbPointInter-1)
Calcul de l'écart moyen entre *les points intermédiaires :*
aint,bint,cint,dint,eint,fint,pint où :
aint[i] = (a[(boucle+1),i] - a[boucle,i]) / NbImage
...
fint[i] = (f[(boucle+1),i] - f[boucle,i]) / NbImage
pint[i] = (p[(boucle+1),i] - p[boucle,i]) / NbImage

Pour Nbim = 1 à NbImage
Choisir un point initial (x,y)

Effectuer l'ajout de l'écart moyen à l'image précédente :
Pour i = 1 à nt
a[boucle,i] = a[boucle,i] + aint[i]
b[boucle,i] = b[boucle,i] + bint[i]
...
p[boucle,i] = p[boucle,i] + pint[i]
Fin i

```
        Pour n = 1 à nbre_point
            choisir un transformation au hasard = t (en fonction des probas)
            déterminer le nouveau point :
                newx = a[boucle,t]*x + b[boucle,t]*y + e[boucle,t]
                newy = c[boucle,t]*x + b[boucle,t]*y + f[boucle,t]
            x = newx, y = newy
            dessiner le point (x,y)
        Fin n
    Fin Nbim
Fin boucle.
```

Les différences essentielles sont donc l'ajout de la boucle sur le nombre de points intermédiaires qui assure le calcul de l'animation entre chacun de ces points et le calcul de l'écart moyen qui s'effectue cette fois entre deux points intermédiaires successifs. L'ajout de cet écart se fait alors toujours sur le point intermédiaire précédent.

Cet algorithme est appliqué à une fougère en trois dimensions dans le programme 'Farn3D' situé en annexe. Le résultat est l'animation de cette fougère avec, en plus des points extrêmes, un point intermédiaire. Une fois de plus, la 'lenteur' relative du calcul nécessite de spécifier un nombre de points assez petit pour obtenir une animation un temps soit peu fluide. La figure 12 ci-dessous reprend les trois points intermédiaires de l'animation (Pt_1, Pt_2 et Pt_3).

Figure 12

La conception d'un IFS en trois dimensions demande la modification de quelques éléments de base. La transformation affine s'exprime alors comme suit :

$$\mathbf{W(x) = Ax + t,}$$

avec

$$W\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & m \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} n \\ q \\ r \end{pmatrix}$$

On utilise donc une troisième composante apportant la troisième dimension. Mais en plus de cela, il faut ajouter des angles ♋, ♌ et ♑. Ils permettent d'obtenir une déformation en trois dimension de l'image (ici de la fougère). Ces angles définissent la projection de l'image de l'objet considéré en trois dimensions sur le plan bidimensionnel que représente une feuille ou l'écran d'un ordinateur. La détermination des coordonnées d'un point sur ce plan à partir de ces coordonnées tridimensionnelles s'obtient grâce aux formules de projections que voici :

$$\mathbf{X' = x \square \cos(б) + y \square \cos(в) + z \square \cos(з)}$$
$$\mathbf{Y' = x \square \sin(б) + y \square \sin(в) + z \square \sin(з)}$$

où (x,y,z) sont les coordonnées du point en 3D. La détermination de ces valeurs peut poser quelques difficultés. Pour mieux comprendre leur implication sur l'image, voyons quelques exemples de variation des valeurs de ces angles.

Pour commencer, nous allons faire varier l'angle ♋ sur quatre valeurs qui sont (15°, 45°, 90°, 135°). Les autres angles ont comme valeur ♌ = 70° et ♑ = 20°.

Figure 13

Nous remarquons que la variation de cet angle implique une rotation de la fougère autour d'un axe placé verticalement au centre de l'image comme sur la figure ci-dessous. Cette impression de rotation est beaucoup mieux rendue lorsque vous calculez et visualisez une petite animation réalisant cela. C'est alors que l'on remarque la rotation de la plante comme indiqué sur la figure 14.

Figure 14

Passons à la variation de ♌. Les valeurs choisies sont (30°, 60°, 100°, 110°). Les valeurs des autres angles sont ♋ = 0° et ♑ = 20°. Voici les images obtenues pour ces valeurs :

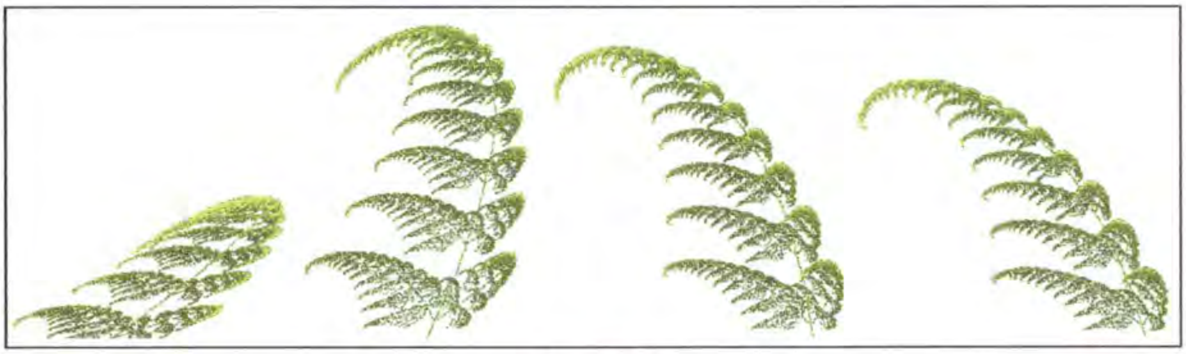

Figure 15

Cette fois, nous pourrions comparer les différences obtenues au déploiement de la plante en partant de la droite (angle aigu) en allant vers la gauche (angle obtus). Le dessin ci-dessous reprend cette variation. Une fois de plus, une petite animation aide à mieux comprendre ces modifications.

Figure 16

Pour terminer, faisons varier l'angle ♑ sur les valeurs (35°, 75°, 100°, 130°) avec ♋ = 0° et ♌ = 90°.

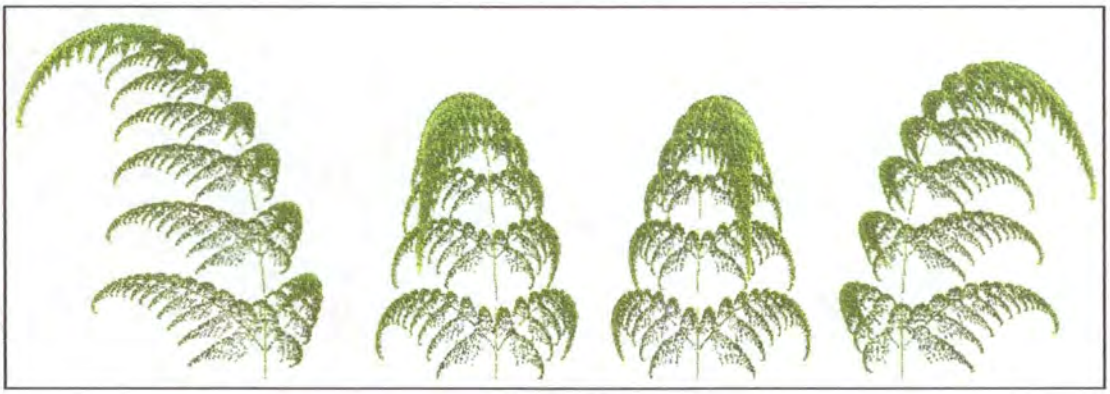

Figure 17

Avec ♑, la fougère se replie sur elle-même et suit en quelque sorte une courbe comme celle reprise ci-dessous.

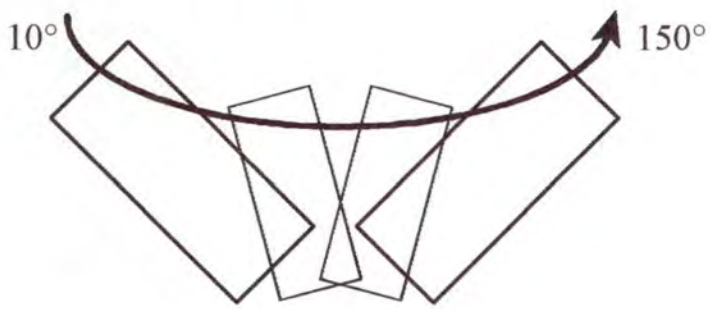

Figure 18

Comme on peut le constater, l'utilisation de ces angles permet la modélisation que l'on souhaite de la plante en trois dimensions. Les compositions possibles sont infinies. Cependant, il est vrai que pour certaines valeurs d'angles, l'image finale n'est pas très représentative. Il faut donc être judicieux dans son choix.

Après le développement de ces deux algorithmes sous Dos, grâce auxquels nous avons pu spécifier la structure générale nécessaire à la réalisation d'animations fractales, il va nous falloir passer à une programmation sous l'environnement Windows et ceci afin de contourner les problèmes que nous rencontrons, à savoir essentiellement une gestion optimale d'images bitmap en mémoire.

Chapitre 2 : Passage à une implémentation sous environnement Windows

Ce chapitre présente l'application réalisée sous Windows pour l'obtention d'animations fractales fluides et de qualité. Nous allons dans un premier temps présenter la conception du programme pour ensuite en venir à sa manipulation dans le prochain chapitre. A cette occasion, nous présenterons quelques exemples d'animations réalisées avec l'application.

Conception :

Pour cette conception, nous allons évidemment conserver les algorithmes mis au point sous Dos. Notre objectif est de pallier aux problèmes que nous avons rencontrés. Tout d'abord, le temps de calcul de chaque image étant beaucoup trop élevé pour obtenir une animation en temps réel, il est nécessaire de calculer à l'avance l'ensemble des images pour ensuite les afficher à l'écran. Il faut alors avoir une méthode de stockage qui permette un transfert rapide des images vers l'écran. Comme nous l'avons déjà vu, la meilleure solution est de les placer en mémoire centrale. C'est là que la programmation sous Windows nous apporte l'élément manquant, la *gestion de la mémoire*. En effet, il nous est maintenant possible de définir une zone mémoire dans laquelle nous allons pouvoir dessiner notre bitmap de l'image fractale souhaitée. Nous pourrons ainsi répéter l'opération le nombre de fois souhaité en fonction du nombre d'images de l'animation et obtenir un stockage de données facile et rapide d'accès. Comme on peut s'en douter, plus le nombre d'images à mémoriser est grand, plus la taille de la mémoire disponible doit être élevée.

Reprenons l'algorithme 4, précédemment défini, et ajoutons-lui les éléments nécessaires pour notre nouvelle stratégie.

Algorithme 5

Initialiser l'ensemble des données :
a,b,c,d,e,f[i,nt]
nt = nbre de transformations
les probas associées = p[i,nt]
NbImage = nbre d'images souhaitées
NbPointInter = nbre de points intermédiaires

Pour boucle = 1 à (NbPointInter-1)
Calcul de l'écart moyen entre les points intermédiaires
Pour Nbim = 1 à NbImage
Choisir un point initial (x,y)
Effectuer l'ajout de l'écart moyen à l'image précédente :
Pour n = 1 à nbre_point
déterminer le point
*dessiner le point sur bitmap[((boucle-1)*NbImage)+Nbim]*
(en mémoire)
Fin n
Fin Nbim
Fin boucle

*Pour Image = 1 à (PointInter-1)*NbImage*
Affichage à l'écran de la bitmap mémoire [Image]
Fin Image.

Cet algorithme garantit cette fois un résultat de qualité suffisante. Voyons maintenant les éléments que nous allons définir dans notre programme.

Il ressort de l'algorithme qu'il faut offrir à l'utilisateur trois grands services au sein du programme :

- la définition et la personnalisation des paramètres de l'IFS (probas...) ainsi que des valeurs de l'animation (nombre d'images, nombre de points intermédiaires...)

- le calcul de l'animation en fonction des valeurs définies
- et la visualisation de l'animation ainsi calculée

Ces trois points sont présents dans l'algorithme dans le même ordre. Nous allons maintenant les étudier indépendamment les uns des autres.

1. <u>Définition des valeurs initiales</u> :

En ce qui concerne les paramètres de transformations et les probabilités, il faut permettre à l'utilisateur de définir, modifier ou effacer l'ensemble des valeurs. Ceci doit être possible pour chacun des points intermédiaires définis. Il faut donc prévoir pour chaque variable un tableau du type *a[1..PointInter, 1..Nbre de transformations]*. Dans notre cas, nous allons choisir de limiter le nombre de points intermédiaires à 5 et le nombre de transformations à 4. Ceci est suffisant pour la modélisation d'animations complexes sur des IFS réalistes. Il nous faudra donc prévoir une boîte de dialogue permettant de modifier l'ensemble de ces valeurs.

Pour les paramètres de l'animation, il est également nécessaire d'avoir une boîte de dialogue pour les modifier. Les valeurs dont nous aurons besoin seront le nombre d'images, le nombre de points intermédiaires utilisés, le nombre de points à calculer pour chaque image. Enfin, nous verrons par la suite que dans certain cas, il est nécessaire de ralentir l'affichage de l'animation issue de la mémoire centrale. A cet effet, nous ajoutons un facteur de ralenti qui permettra d'obtenir le ralentissement souhaité lors de l'affichage de la séquence animée.

Remarquons que le nombre d'images que nous définissons correspond en fait au nombre d'images calculées *entre chaque point intermédiaire*. Ceci découle immédiatement de l'algorithme 5. Il faudra donc y veiller ultérieurement afin de ne pas définir trop d'images à calculer. En effet une animation avec 5 points intermédiaires et 25 images impliquerait une animation de 100 images, ce qui représente une place mémoire considérable.

2. Calcul de l'animation :

Il suffit ici de prévoir un bouton de lancement du calcul dans l'écran principal. Son déclenchement impliquera le calcul de l'animation avec les paramètres définis à ce stade. L'ensemble de l'animation sera stocké en mémoire.

Il serait bon d'afficher également à l'écran le calcul de chaque image au fur et à mesure afin de permettre à l'utilisateur de se faire une idée du résultat et éventuellement d'arrêter le calcul si cela ne correspond pas à son souhait.

Les images sont calculées selon le principe habituel. Cependant, l'environnement Windows nous offre la possibilité de manipuler des couleurs sur 24 bits, ce qui donne une palette de plus de 16 millions de couleurs. Il serait donc positif pour le réalisme de l'animation de prévoir une coloration des plantes. En étudiant à cette fin le comportement du point courant sur l'image pour dessiner l'attracteur de l'IFS, on se rend compte qu'il parcourt plus souvent certaines parties de l'image. Ainsi, on peut en déduire que certaines transformations ont plus de *'poids"* que d'autres. Ces poids sont en fait dépendants des probabilités définies pour les transformations de l'IFS. On va donc considérer une couleur différente en fonction du poids de chaque élément de l'image. Les éléments sont en fait les pixels de la bitmap.

Pour colorier l'image, il suffit donc, avant de mettre un point, de considérer la couleur du point existant au préalable aux mêmes coordonnées. Ensuite, faire varier légèrement la teinte de ce point. On peut ainsi définir une variante sur une palette de couleur.

A titre d'exemple, la couleur de base de la fougère est le vert foncé. Plus on se dirige vers les extrémités des feuilles, plus le poids de chaque pixel est élevé. Ceci nous suggère alors de faire diminuer la teinte verte en fonction de ce poids. Le résultat est une fougère coloriée par un dégradé du vert foncé au vert clair en partant de la tige vers les extrémités des feuilles (cf. dessins couleurs présentés précédemment lors de l'étude des angles sur la fougère 3D). Il est évident qu'une palette différente est nécessaire pour d'autres objets.

Voici le complément à apporter à l'algorithme pour réaliser ce qui vient d'être dit :

Définition des coordonnées du point → (x,y)

Prendre couleur de (x,y) existant → coul
Faire varier coul → (ex.: coul = coul + 15)
Dessiner le point en (x,y) avec comme couleur coul.

3. Lancement de l'animation :

Une fois de plus, un simple bouton placé dans l'écran principal sera suffisant pour effectuer l'affichage de l'ensemble de la séquence à l'écran. La pression sur ce bouton impliquera le transfert des bitmaps précalculées et placées en mémoire vers l'écran.

En résumé, notre écran principal devra comporter l'affichage des paramètres de l'animation (nombre d'images, nombre de points par image, facteur de ralenti, nombre de points intermédiaires) ainsi qu'un bouton d'appel d'une boîte de dialogue permettant de modifier ces valeurs. Il faudra également y définir un bouton pour le calcul de l'animation et un bouton pour le lancement de la séquence. Bien sûr, nous y placerons un menu reprenant au moins les fonctions ouverture et sauvegarde de fichiers ainsi que la possibilité de quitter le programme. Pour terminer, n'oublions pas l'élément principal, qui est la fenêtre dans laquelle nous dessinerons les bitmaps et nous afficherons les séquences. A ce propos, ses dimensions seront fixées à 300 x 300 points de côté, et ceci afin de ne pas définir une trop grande fenêtre qui impliquerait une réservation d'espace mémoire excessive. En effet, une image de 300 x 300 points comporte donc 90000 points, chaque point comportant une composante sur 24 bits, ce qui nous donne 270 Kilobytes par image mémorisée. Nous obtenons donc une réservation de place mémoire de l'ordre de 3 Mégabytes par seconde d'animation (soit 10-11 images par seconde) ou encore 15 Mégabytes pour 5 secondes. Une fenêtre de 300 points de côté semble donc être un maximum.

Ayant défini les éléments principaux nécessaires à la réalisation d'animations, passons maintenant à une étude du programme réalisé en Turbo Pascal sous Windows et qui est repris en annexe. Il s'agit du programme ***'Fractwin'***. Ce programme reprend les éléments de base définis ci-dessus et s'est vu également adjoindre un certain nombre de compléments ayant comme objectif principal de faciliter la manipulation de l'application.

Chapitre 3 : Manipulation du programme sous Windows et exemples de séquences animées obtenues

Pour apprendre la manipulation de l'application, nous allons étudier en détail chacun de ses composants. A cette fin, nous présenterons chaque écran et boîte de dialogue et nous expliquerons la raison et le fonctionnement des divers éléments s'y trouvant. Dans un second temps, nous exploiterons l'application pour en obtenir trois séquences animées et, par la même occasion, pour tester les performances du programme.

Ecran principal :

Il s'agit de l'écran que l'on rencontre d'entrée de jeu lorsque l'application est exécutée. Il est repris dans la figure 19 ci-dessous :

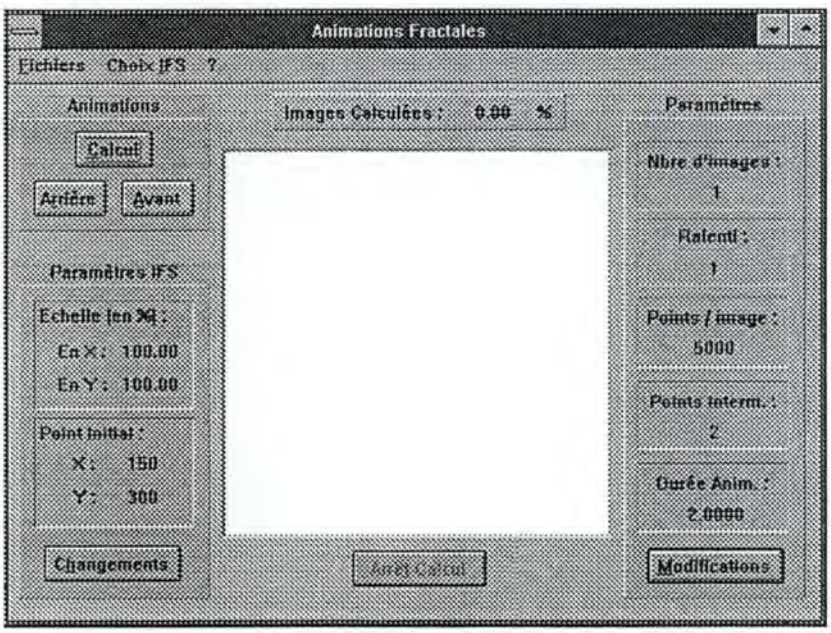

Figure 19

Analysons les différents composants. Tout d'abord, le menu comporte trois éléments principaux qui sont *'Fichiers'*, *'Choix IFS'* et *'?'*. Le sous-menu *'Fichiers'* contient l'ouverture, la sauvegarde ainsi que la mise à zéro d'un fichier sur disque. Il

contient également l'option *'Quitter'* permettant de sortir du programme. Le second élément du menu principal est *'Choix IFS'*. Ce menu est proposé afin de faciliter la manipulation des concepts d'animation et d'IFS pour des personnes novices en la matière. En effet, il initialise par défaut divers types d'animations prédéfinies. Ceci permet d'obtenir un ensemble de données de base permettant de se familiariser avec la manipulation du programme. Le sous-menu contient le choix entre trois animations : la croissance d'une fougère en 2D (*'Farn2D'*), le mouvement de gauche à droite d'une fougère 3D (*'Farn3D'*) et la croissance d'un arbre (*'Arbre'*). Deux de ces animations seront étudiées et utilisées dans la seconde partie de ce chapitre. Enfin, le choix de *'?'* vous permettra d'en savoir un peu plus sur l'application que vous êtes en train d'utiliser...

Le reste de la fenêtre, situé sous cette barre des menus, est constitué d'une boîte de dialogue qui contient tous les champs d'édition et boutons ainsi que la bitmap. Analysons ces éléments un à un.

Nous avons en fait trois blocs principaux. Ils portent les noms de : *'Animations'*, *'Paramètres IFS'* et *'Paramètres'*.

Le premier comporte trois boutons qui permettent l'accès pour l'utilisateur aux deux services de base, à savoir le calcul de l'animation et le lancement de l'animation. Une fois activé, le bouton *'Calcul'* va enclencher le calcul de l'animation définie par les paramètres enregistrés. A ce moment, les autres éléments vont "se griser" afin d'empêcher l'utilisateur de modifier des paramètres nécessaires au calcul. Immédiatement après, le bouton *'Arrêt Calcul'* va se dégriser afin de permettre à l'utilisateur de stopper le calcul d'animation en cours. Notons également qu'au fur et à mesure de l'avancement du calcul, le pourcentage d'images calculées est affiché dans le champ statique situé dans la partie supérieure de l'écran à l'arrière de l'intitulé *'Images Calculées'*. Lorsqu'un calcul d'animation a été effectué, les images sont alors stockées en mémoire. Le lancement de l'animation s'effectue à l'aide des boutons 'Arrière' et 'Avant'. Dans le premier cas, l'animation va se dérouler dans le sens inverse de celui de son calcul, c'est-à-dire de la dernière image à la première, tandis que dans le second cas, la visualisation se fera de la première à la dernière image.

Le second bloc, intitulé 'Paramètres IFS', contient quelques caractéristiques concernant le dessin de l'attracteur de l'IFS. Il s'agit de champs d'édition reprenant

les coordonnées du point initial du dessin de l'attracteur dans la bitmap en (X,Y) et l'échelle appliquée à cet attracteur lors du dessin sur la bitmap. Le point initial représente le point à la base de l'image (dans le cas de la fougère ou de l'arbre, c'est la base de la tige). L'échelle, d'une valeur maximum de 100 %, permet de diminuer la taille de l'attracteur si celui-ci est trop grand, ou encore d'apporter un écrasement de l'image en ordonnées ou en abscisses. Les champs étant statiques, l'utilisateur qui souhaite modifier ces valeurs, peut le faire en enfonçant le bouton *'Changements'* qui appellera aussitôt la boîte de dialogue *'Paramètres IFS'* que nous décrirons plus loin. Ces valeurs ont été ajoutées pour permettre de reparamétrer une image qui sortirait de la bitmap ou qui serait trop grande.

Le troisième bloc est celui des *'Paramètres'*. Il s'agit en fait des paramètres de l'animation. Nous y retrouvons le nombre d'images, le facteur de ralenti, le nombre de points par image, le nombre de points intermédiaires et une estimation de la durée de l'animation. Rappelons que ce nombre d'images correspond au nombre d'images entre chaque point intermédiaire. En ce qui concerne la durée de l'animation, l'estimation qui en est faite n'est guère représentative. En fait, il est difficile de tenir compte de tous les facteurs perturbateurs de cette durée (matériel d'affichage lent, mémoires lentes, mémoires insuffisantes...). Il est également possible de changer ces valeurs et pour ce faire il suffit d'enfoncer le bouton *'Modifications'* qui appellera la boîte de dialogue *'Paramètres animations'* que nous allons étudier ci-après.

Il reste enfin un petit mot à dire sur la fenêtre blanche située au centre de l'écran. Il s'agit de la fenêtre fixe que nous avons définie pour l'affichage de nos animations. Elle fait exactement 300 x 300 points comme nous l'avions déjà précisé. Nous y projetterons l'ensemble des images calculées et stockées en mémoire lors du lancement de l'animation et nous y afficherons également le dessin de chaque image au fur et à mesure de l'avancement du calcul. Ceci permet à l'utilisateur de se faire une idée du résultat et éventuellement de stopper le calcul si il le souhaite.

Après la présentation de la fenêtre principale, passons à l'analyse des boîtes de dialogues auxiliaires. Ces boîtes sont au nombre de trois.

Boîte de dialogue 'Paramètres animations' :

L'appel de cette boîte de dialogue se fait en poussant sur le bouton *'Modifications'* du bloc *'Paramètres'*. On obtient ainsi, à l'écran, l'équivalent de la figure 20 suivante :

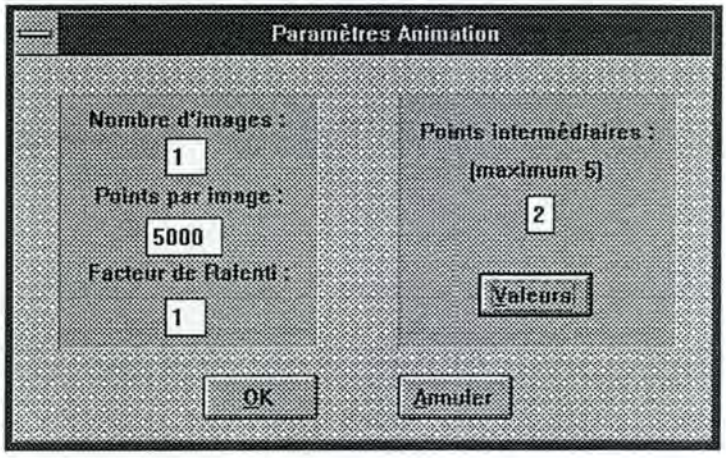

Figure 20

Il est alors possible par l'intermédiaire des champs d'édition de modifier ce qui ne convient pas :

- le nombre d'images (compris entre 1 et 99)
- le nombre de points par image (maximum 99999)
- le facteur de ralenti (1 par défaut)
- le nombre de points intermédiaires (minimum 2 et maximum 5)

Comme pour toute boîte de dialogue de ce type, le bouton 'OK' enregistre les modifications éventuelles et replace l'utilisateur dans la fenêtre parent (l'écran principal) tandis que le bouton 'Annuler' ne tient pas compte des changements et renvoie également dans la fenêtre parent.

On peut constater la présence d'un bouton *'Valeur'* situé au-dessous du champs d'édition des points intermédiaires. Il appelle la boîte de dialogue 'Points Intermédiaires' qui permettra de définir, modifier ou encoder les paramètres et probabilités des IFS situés à chaque point intermédiaire. C'est cet écran que nous allons maintenant étudier.

Boîte de dialogue 'Points Intermédiaires' :

Après avoir pressé le bouton *'Valeur'*, on obtient donc la boîte de dialogue de la figure 21 ci-dessous.

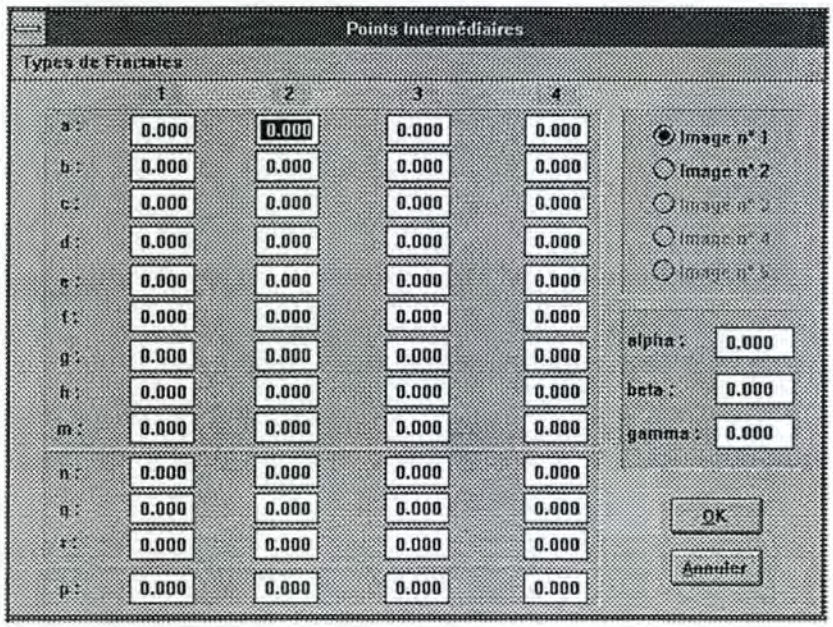

Figure 21

Cette boîte de dialogue est bien chargée. Il a fallu en effet y reprendre l'ensemble des paramètres, des angles ainsi que les probabilités définissant un IFS avec un maximum de quatre transformations. On retrouve donc dans la boîte un certain nombre de champs d'édition permettant la modification de ces valeurs. Pour savoir quelle valeur il faut changer, il convient de lire le tout comme un tableau en considérant ligne et puis colonne. Par exemple, le premier champ d'édition correspond à la valeur de a (en ligne) [1] (en colonne). La dernière ligne reprend de cette façon les probabilités liées à chaque transformation (1 à 4, de gauche à droite). Enfin, à droite de l'écran, sont repris les angles α, β et γ de l'IFS.

Si on ne souhaite pas introduire l'ensemble des valeurs pour un IFS particulier, il est possible d'utiliser des paramètres prédéfinis et ceci à l'aide du menu. Le sous-menu contient les termes suivants : *'Mise à zéro'*, *'Farn'* (soit 2D, soit 3D Plié Gauche, soit 3D Normal, soit 3D Plié Droite) et *'Arbre'*. La sélection d'un de ces éléments donnera immédiatement l'ensemble des paramètres par défaut

le définissant. Ainsi, choisir 'Arbre' implique immédiatement le passage des valeurs des champs d'édition aux valeurs correspondant à l'arbre IFS. De même le choix de la mise à zéro impliquera la réinitialisation des valeurs de l'écran. Une fois de plus, cette option a été ajoutée en vue de faciliter la manipulation du programme par une personne novice en matière d'IFS.

Comme nous l'avons déjà expliqué, il est possible de spécifier des valeurs différentes pour l'IFS à chaque point intermédiaire. A cette fin, nous avons ajouté une sélection du point intermédiaire pour lequel les paramètres sont à définir dans le coin supérieur droit de la boîte de dialogue. En sélectionnant par exemple *'Image n°2'*, on peut modifier les valeurs des paramètres du point intermédiaire n° 2. Les points intermédiaires qui ne sont pas accessibles sont grisés. En fait, on ne peut définir que le nombre de points intermédiaires que l'on a spécifié dans la boîte de dialogue *'Paramètres Animation'*. Si on définit quatre points intermédiaires, automatiquement *'Image n°5'* sera grisé dans la boîte *'Points Intermédiaires'*.

Il est à noter que dans ce cas-ci, la pression du bouton *'OK'* implique la sauvegarde des modifications apportées aux valeurs visibles à l'écran mais également des modifications réalisées sur les autres points intermédiaires. Le bouton *'Annuler'* quant à lui renvoie toujours à la fenêtre parent sans tenir compte de quelque modification que ce soit.

Boîte de dialogue 'Paramètres IFS' :

Cette dernière boîte permet la correction de l'échelle de l'image de l'attracteur ainsi que le point initial. Les champs d'édition permettent une fois de plus de modifier les valeurs de l'échelle en X ou en Y et les coordonnées du point situé à la base de la tige de la plante dessinée. Ces valeurs se répercutent sur le résultat au calcul de l'animation et non pas sur les images précalculées et qui sont affichées lors du lancement de l'animation. La figure 22 représente cette boîte de dialogue.

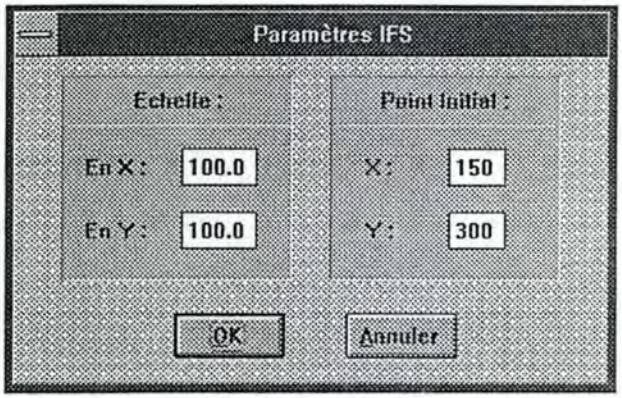

Figure 22

Après l'analyse des éléments composant le programme, nous allons pouvoir l'utiliser en exécutant le calcul d'animation de quelques séquences. Ceci nous permettra par la même occasion de présenter les résultats que nous pouvons obtenir avec cette application.

Exemples d'animations obtenues avec le programme :

Ces exemples sont au nombre de deux avec en plus un cas particulier que nous présenterons par après. Les animations que nous avons choisies de présenter ici sont : 'Le mouvement de gauche à droite d'une fougère en 3D' et 'La croissance d'un arbre'. Cette dernière consiste en un passage d'un point initial à un point final, tandis que la première comporte un point intermédiaire supplémentaire afin de mettre en application ce que nous avons étudié.

La croissance d'un arbre :

Il s'agit de la dernière des animations prédéfinies dans le programme et que l'on obtient en choisissant l'option *'Arbre'* du sous-menu de *'Choix IFS'*. Le point intermédiaire 1 (point initial) aura ses paramètres à zéro pour signifier que l'on part de rien. Remarquons au passage que les probabilités, quant à elles, ne sont pas nulles afin d'assurer un passage de l'IFS 'nul' (c'est-à-dire IFS avec paramètres nuls) à l'arbre le plus fluide possible. Il en est de même pour les angles. Enfin, le point intermédiaire 2 (point final) contiendra les valeurs de l'IFS 'Arbre'.

Le nombre de points choisi pour le calcul d'une image est 80000 et le nombre d'images est 12. L'échelle est de 100 % en X et Y et le point de la base de la tige est au centre dans le fond de la bitmap. Le calcul de l'animation est ensuite lancé et l'attente commence... Comme nous l'avons déjà expliqué, le temps de calcul est considérable. Mais nous reviendrons à ce problème un peu plus loin.

Le résultat donne un ensemble de 12 images qui sont reprises dans la partie supérieure de la page 57. Evidemment, ces planches ne rendent pas une idée exacte de l'animation. Cependant, nous pouvons remarquer que les changements sur l'attracteur s'opèrent harmonieusement.

Le mouvement d'une fougère 3D :

Il s'agit de la seconde animation prédéfinie dans le programme. Elle est obtenue en sélectionnant l'option *'Farn 3D'* dans le sous-menu de *'Choix IFS'*. L'animation se compose de trois points intermédiaires. Le premier définit une

fougère 3D pliée sur la gauche, le second définit une fougère 3D normale et le troisième définit la même fougère mais pliée sur la droite.

Le nombre de points par image est 80000 et le nombre d'images est de 10. Attention, ce nombre d'images correspond aux images situées entre deux points intermédiaires. Cela nous donne donc ici 20 images en tout pour l'animation. L'échelle est de 90 % en X et Y et le point de la base de la tige est légèrement décalé vers la droite pour mieux centrer l'image. Le calcul de l'animation est ensuite lancé et l'attente recommence...

Le résultat donne un ensemble de 20 images qui sont également reprises à la page 56. Si on analyse les paramètres définis aux points intermédiaires, on constate que les seules modifications nécessaires pour obtenir ces images sont apportées aux angles.

Un cas particulier d'animation : 'le morphing' :

Nous ne pouvions pas ignorer cet aspect particulier des animations qu'est le morphing entre des images. Il s'agit d'un terme anglais englobant les phénomènes d'animations qui réalisent le passage d'une forme à une autre par l'intermédiaire d'une animation fluide. Cet effet est devenu fort à la mode depuis quelques années dans les montages à effets visuels. Certaines publicités ont déjà utilisé le principe pour transformer par exemple une voiture en félin ou un escargot en grenouille ou encore pour réaliser des déformations particulières. Le programme que nous avons développé n'était pas prévu pour la réalisation de ce type d'animation. Cependant sa conception et son architecture permettent l'obtention d'animations 'morphing'. Il suffit pour cela de définir un IFS au point initial (point intermédiaire 1) et d'en définir un autre au point final. Le résultat de l'animation sera le passage continu entre le premier attracteur et le second.

Pour mieux comprendre le phénomène, nous allons le reprendre dans deux exemples. Le premier est la séquence de passage de la courbe de Koch à la fougère, le tout en 9 images. Le résultat est présenté dans la figure 23 ci-dessous.

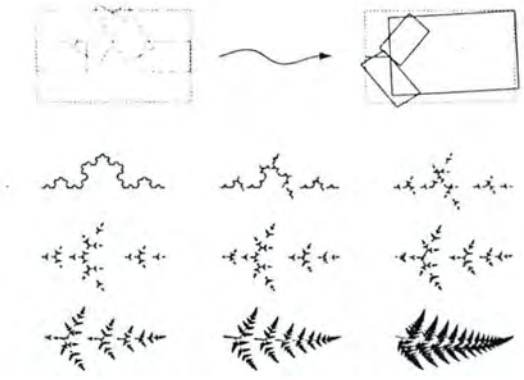

Figure 23

Une autre séquence a été réalisée avec notre programme. L'IFS de départ est la fougère 3D et l'IFS final est l'arbre. Il s'agit des paramètres que nous avons repris dans le menu des différents IFS. Nous retrouvons également un passage continu et fluide entre ces deux IFS. Le résultat est repris dans la partie inférieure de la page 57 en 8 images.

Image 1

Image 2

Image 3

Image 4

Image 5

Image 6

Image 7

Image 8

Image 9

Image 10

Image 11

Image 12

Image 13

Image 14

Image 15

Image 16

Image 17

Image 18

Image 19

Image 20

Image 1 Image 2 Image 3 Image 4 Image 5

Image 6 Image 7 Image 8 Image 9

Image 10 Image 11 Image 12

1 2 3 4

5 6 7 8

Tests du programme sur diverses machines :

Dans le but de familiariser l'utilisateur avec l'attente durant le calcul, nous allons reprendre ici différents tests effectués sur deux machines différentes et nous allons présenter ensuite les délais d'attente nécessaires à l'obtention des animations correspondantes. Nous avons en fait chronométré le temps de calcul des trois animations que nous venons de présenter ci-dessus. Mais avant de les analyser, voyons les configurations de PC sur lesquels les tests ont été effectués.

Machine n° 1 :

La première machine est un 486 DX-50 Mhz avec une mémoire centrale de 16 Mégabytes et une mémoire cache externe de 256 Kilobytes. Le disque dur est de 350 Mégabytes avec un temps d'accès moyen inférieur à 14 millisecondes et un taux de transfert de l'ordre de 1 Mégabyte par seconde. La carte graphique est de type Tseng Labs avec 1 Mégabyte de mémoire permettant un affichage simultané de quelque 32768 couleurs, l'écran étant évidemment un SVGA couleur. Les performances de ce PC sont évaluées aux alentours de 40 Mips. Enfin, nous avions défini sous Windows une mémoire virtuelle sur disque dur d'environ 10 Mégabytes. Notons au passage que cette mémoire secondaire n'intervient que lorsque la mémoire centrale arrive à saturation. Il peut par exemple arriver que durant le temps de calcul, certaines images soient transférées sur disque dur. Lors de la première visualisation de l'animation, ces images seront retransférées en mémoire, ralentissant l'affichage momentanément. Cependant, les visualisations suivantes seront normales et fluides.

Machine n° 2 :

La second PC est un 386 DX-25 Mhz avec une mémoire centrale de 4 Mégabytes et aucune mémoire cache externe. Le disque dur est de 120 Mégabytes avec un temps d'accès moyen de 16 millisecondes et un taux de transfert de l'ordre de 800 Kilobytes par seconde. La carte graphique est une Paradise 512 Kilobytes permettant d'afficher 16 ou 256 couleurs suivant la résolution, l'écran étant toujours un SVGA couleur. La mémoire virtuelle sur

disque définie pour Windows est d'environ 5 Mégabytes. Pour cette configuration, cette mémoire peut être souvent sollicitée pour des animations d'une certaine taille. Le résultat sera une saccade lors de la visualisation de l'animation.

Le tableau ci-dessous reprend les temps obtenus. Signalons qu'il s'agit de temps chronométrés manuellement, ils sont présentés ici à titre indicatif uniquement et ne sont probablement pas d'une précision extrême.

	486 DX-50 Mhz		386 DX-25 Mhz	
	Tps pour 1 image	Tps pour tout	Tps pour 1 image	Tps pour tout
Farn 3D (20 images)	31"	10' 40"	9' 30"	3h 10'
Arbre (12 images)	28"	5' 30"	8' 35"	1h 42' 25"
Morph (8 images)	32"	4' 24"	8' 42"	1h 9' 40"

La carte graphique de la première machine donne un résultat tout à fait acceptable quant au dégradé de couleur sur les images. A l'inverse, une carte n'offrant que 256 couleurs en même temps à l'écran donne un résultat plutôt médiocre pour les dégradés. Il est donc préférable d'utiliser l'application avec des cartes graphiques performantes.

Notons enfin qu'avec la première machine, une mémoire de 16 Mégabytes offre la possibilité de stocker environ 60 images (270 Kilobytes pour une images) tandis qu'avec 4 Mégabytes, nous sommes limités à plus ou moins 15 images.

Comme on peut le constater, la configuration d'une machine pour utiliser ce programme doit être musclée. Le PC idéal dans cette situation est un 486 DX2-66 Mhz avec 16 ou 32 Mégabytes de mémoire et une carte vidéo permettant l'affichage simultané de 32768, voir 16 millions de couleurs.

Conclusion

Cette partie pratique nous a permis d'appliquer, dans le cadre de trois programmes, les notions étudiées dans le première partie. Nous avons réalisé l'application souhaitée sans pour autant pouvoir la développer suffisamment afin de mettre en pratique l'ensemble des notions que nous avions rencontrées. Ceci est essentiellement dû aux problèmes rencontrés relativement aux configurations puissantes requises pour les machines gérant le programme. Le problème principal survenu fut celui de la place nécessaire au stockage des images tout en permettant un accès rapide pour l'obtention de mouvements fluides.

Il nous a cependant été possible d'aborder le problème pour en faire ressortir ces limites et, malgré tout, d'arriver à réaliser un programme permettant le calcul de petites animations assez réalistes. Ainsi les notions d'IFS, d'algorithmes-chaos, de playback en temps réel, de positions clés (ou points intermédiaires) et de coloration d'attracteurs ont pu être mises en application au sein du programme développé pour l'environnement Windows.

Nous pouvons donc conclure que, malgré la nécessité d'utiliser des machines puissantes pour les calculs, la réalisation d'animations fractales réalistes, quelles qu'elles soient, est possible et réalisable sur un ordinateur personnel.

CONCLUSION GENERALE

III. Conclusion générale

Les fractales sont des objets mathématiques merveilleux et fascinants. Elles sont un régal pour les yeux et on ne se lasse pas d'en admirer les formes et les couleurs. De plus, elles étonnent de par leur nombre et de par la quantité inimaginable de variantes possibles à partir d'un même objet.

Notre objectif était de rendre plus attrayant encore cet univers extraordinaire en lui donnant un aspect vivant, en lui permettant de se mouvoir et de prendre vie. Il nous a été possible d'évoluer quelque peu dans cette direction. En effet, la réalisation de notre application a permis de nous apprendre comment produire des animations d'objets fractals. Nous nous sommes limités aux plantes pour les raisons déjà évoquées, mais l'application de déformations ou de mouvements à tout autre type d'objet fractal peut être obtenue suivant les mêmes principes. Pour aboutir à ce résultat, il nous a fallu parcourir les théories sur les IFS et leurs algorithmes de modélisation d'attracteurs ainsi que sur les notions de base des séquences animées. Les divers éléments ressortant de cette analyse nous ont ensuite permis de réaliser les programmes souhaités.

Cependant, la remarque essentielle à préciser dans cette conclusion reste l'aspect "configuration machine". Comme nous avons pu nous en rendre compte, le temps de calcul requis et la place mémoire nécessaire sont les facteurs clés qui nous ont guidé tout au long de la réalisation de ce travail. Il faut, hélas, constater que la simulation de mouvement, pour des objets fractals réalistes, en temps réel sur un ordinateur personnel ne nous semble pas pour aujourd'hui, ni pour demain d'ailleurs. En effet, malgré la recherche d'optimisation du calcul de l'attracteur et de toute l'animation, il faudrait avoir recours à d'autres types de hardware pour atteindre la réalisation de ces animations en temps réel.

Enfin, il serait intéressant d'étoffer l'application obtenue en apportant des possibilités supplémentaires à l'utilisateur. La gestion simultanée de plusieurs objets dans l'animation, l'introduction de séquences sonores, l'évolution vers des animations de plus grandes tailles, la mise au point de techniques de compression

des images en vue d'optimiser la place mémoire utilisée et éventuellement d'avoir recours à une unité de sauvegarde auxiliaire sont autant d'aspects qui apporteraient un plus pour la réalisation et la manipulation de ces animations. Ceci pourrait d'ailleurs être l'objet d'un autre mémoire...

Nous voici donc arrivé à l'issue de ce mémoire dont la réalisation fut, pour ma part, enrichissante et intéressante. Il aurait certes fallu plus de temps pour approfondir les recherches et aboutir à une application plus puissante. Cependant, si ce travail a donné au lecteur le goût de cette nouvelle dimension apportée à l'univers fractal, nous l'invitons à consulter les ouvrages référencés dans la bibliographie à la page suivante. Il y trouvera de quoi satisfaire sa curiosité et affiner ses connaissances du sujet.

III. Bibliographie

- "Fractals for the classroom" (Part One : Intro to Fractal and chaos), H.O. Peitgen, H. Jürgens, D. Saupe. National Council of Teachers of Mathematics. New York, Paris, 1991.

- "The science of fractal images", H.O. Peitgen, D. Saupe. Springer-Verlag. New York Inc., 1988.

- "Chaos and fractals" (Proceedings of Symposia in applied mathematics, volume 39), K.T. Alligood, J.A. Yorke, M.F. Barnsley, B. Branner, J. Harrison, P.J. Holmes, American mathematical society. Etats-Unis, 1989.

- "Chaos and fractals : New frontiers of science", H.O. Peitgen, H. Jürgens, D. Saupe. Springer-Verlag. New York, Paris, 1992.

- "Computer animation theory and practice", N. Magnerrat-Thalmann, D. Thalmann. Springer-Verlag. Tokyo, 1985.

- "Fractal : Programming in Turbo Pascal", R. T. Stevens. M&T Publising.

- "Dictionnaire du cinéma", J-L. Passek. Edition Larousse. Canada, 1986.

- "PC Expert", Numéro 14. France, Mai 1993.

- "L'encyclopédie de A à Z", Editions Erasme. Bruxelles, 1979.

- "Turbo Pascal pour Windows, le livre officiel Borland", T. Swan, Sybex. Paris, 1992.

- "Génération d'images fractales et implémentation en langage objet", W. Heinen, Mémoire présenté en vue de l'obtention du titre de licencié et maître en inofrmatique. Namur, 1992.

ANNEXES

PLAN

Les différentes parties de cette annexe sont présentées ci-dessous :

Annexe A : Listing du programme "*Farn2D*" en Turbo Pascal. Il s'agit d'un programme d'animation sous Dos permettant d'obtenir la croissance d'une fougère. L'animation est simplement composée d'un point initial et d'un point final.

Annexe B : Listing du programme "*Farn3D*" en Turbo Pascal. Il s'agit d'un programme d'animation sous Dos permettant d'obtenir un mouvement de balancement de gauche à droite pour une fougère en trois dimensions. Cette fois, l'animation est d'un point initial, d'un point final et d'un point intermédiaire supplémentaire.

Annexe C : Listing du programme principal de ce mémoire réalisé en Turbo Pascal sous Windows 3.1. Il s'agit du programme "*Fractwin*". Ces objectifs et son utilisation ont été présentés dans la partie pratique.

Vous trouverez en fin d'annexe une disquette contenant l'ensemble de ces listings tapés ainsi que les exécutables prêts à l'utilisation et répartis dans deux répertoires "*Dos*" et "*Windows*".

Annexe A : Programme "*Farn2D*"

```
program farn2d;

{*-------------------------------------------------------------*}
{*  Ce programme réalise une petite animation d'une feuille  *}
{* de fougère en 2D et ceci concernant sa croissance. Partant*}
{* d'un point, cette feuille va grandir vers sa forme        *}
{* définitive.                                               *}
{*                                                           *}
{*   Pour se faire, il va vous falloir introduire le nombre  *}
{* d'images souhaitées dans l'animation ainsi que le nombre  *}
{* de points                                                 *}
{* calculés par images (respectivement nbre et point).       *}
{*-------------------------------------------------------------*}

  uses crt,dos,graph;
  label 1;


{*****************************************}
{*   Déclaration des variables utilisées   *}
{*****************************************}

  var a,b,c,d,e,f,p      : array[1..4] of real;
      ap,bp,cp,dp,ep,fp  : array[1..4] of real;
      as,bs,cs,ds,es,fs  : array[1..4] of real;
      aux                : char;
      gm,gp,i,t,boucle,nbre,taille,point  : integer;
      nx,ny,x,y,trans    : real;



{**************************}
{*   Programme principal   *}
{**************************}

  begin

  {*   Initialisation des variables contenant les paramètres *}
  {*   de l'IFS final                                         *}

       a[1]:=-0.15; b[1]:= 0.28;   e[1]:= 0;
       c[1]:= 0.26; d[1]:= 0.24;   f[1]:=20;   p[1]:=0.09;
       a[2]:= 0.85; b[2]:= 0.04;   e[2]:= 0;
       c[2]:=-0.04; d[2]:= 0.85;   f[2]:=72;   p[2]:=0.80;
       a[3]:= 0.20; b[3]:=-0.21;   e[3]:= 0;
       c[3]:= 0.18; d[3]:= 0.22;   f[3]:=72;   p[3]:=0.09;
       a[4]:= 0.00; b[4]:= 0.00;   e[4]:= 0;
       c[4]:= 0.00; d[4]:= 0.16;   f[4]:= 0;   p[4]:=0.02;

  {*   Demande à l'utilisateur de donner le nombre d'images   *}
```

```
  {*   souhaitées ainsi que le nombre de points calculés      *}
  {*   par image                                              *}

       clrscr;
       writeln ('Veuillez entrer le nombre d''images souhaitées.');
       readln (nbre);

       writeln;
       writeln ('Veuillez entrer le nombre de points pour chaque
image (1000 … 10000).');
       readln (point);

  {*   Initialisation des variables contenant les paramètres  *}
  {*   de l'IFS initial                                       *}

       for i:=1 to 4 do begin

          ap[i]:=0;bp[i]:=0;cp[i]:=0;dp[i]:=0;ep[i]:=0;fp[i]:=0;

       end;

  {*   Calcul de l'écart moyen défini entre l'IFS initial et    *}
  {*   final et ceci en fonction de nombre d'images définies    *}
  {*   pour l'animation                                         *}

       for i:=1 to 4 do begin

          as[i]:=(a[i]-ap[i])/nbre;
          bs[i]:=(b[i]-bp[i])/nbre;
          cs[i]:=(c[i]-cp[i])/nbre;
          ds[i]:=(d[i]-dp[i])/nbre;
          es[i]:=(e[i]-ep[i])/nbre;
          fs[i]:=(f[i]-fp[i])/nbre;

       end;

  {*   Initialisation de l'écran graphique pour l'affichage    *}
  {*   de l'attracteur                                         *}

       gp := 5;
       gm := 3;
       initgraph(gp,gm,'');
       if graphresult< grOK then
          halt(1);
       randomize;

       SetVisualPage (0);


  {*   Début de la boucle principale exécutant le calcul des     *}
  {*   images de 1 à nbre                                       *}

       for boucle:=1 to nbre do begin
```

```
{*   Ajout des écarts moyens à l'ensemble des paramètres    *}
{*   de l'IFS                                               *}

     for i:=1 to 4 do begin

        ap[i]:=ap[i]+as[i];
        bp[i]:=bp[i]+bs[i];
        cp[i]:=cp[i]+cs[i];
        dp[i]:=dp[i]+ds[i];
        ep[i]:=ep[i]+es[i];
        fp[i]:=fp[i]+fs[i];

     end;

     SetActivePage (boucle);
     clearviewport;
     y:=10;x:=10;

{*   Pour chaque image, calcul de tous les points à afficher  *}
{*   selon l'algorithme que nous avons défini dans la théorie *}
{*   et affichage de ces points à l'écran                     *}

     for i:=1 to point do begin

         if i>10 then
         putpixel(trunc(x)+250,350-trunc(y/2),10);

         trans := random;
         if trans <= p[1] then begin t:=1; goto 1; end;
         if trans <= (p[1]+p[2]) then begin t:=2; goto 1; end;
         if trans <= (p[1]+p[2]+p[3]) then begin t:=3; goto 1; end
                                      else t:=4;

1:       nx := (ap[t]*(x-10)+bp[t]*y+ep[t]+10);
         ny := (cp[t]*(x-10)+dp[t]*y+fp[t]);
         x:=nx;
         y:=ny;

     end;
     setvisualpage (boucle);

     end;

 readln(aux);
 closegraph;
end.
```

Annexe B : Programme "*Farn3D*"

```
program farn3d;


{*-------------------------------------------------------------*}
{*  Ce programme réalise une petite animation d'une feuille de  *}
{* fougère en 3D. Son mouvement sera un mouvement de balancement*}
{* de gauche à droite et ceci en passant par un point inter-    *}
{* médiaire en plus des points initial et final.               *}
{*                                                             *}
{*   Pour se faire, il va falloir introduire le nombre d'images *}
{* souhaitées dans l'animation ainsi que le nombre de points    *}
{* calculés par images (nbre et point)                         *}
{*-------------------------------------------------------------*}


uses CRT,Graph,Fractal;


{*****************************************}
{*   Déclaration des constantes utilisées   *}
{*****************************************}


const

     {*   Déclaration des angles de projections des images   *}
     {*   de l'animation                                      *}

     alpha: array[0..2] of real = (30,45,15);
     beta: array[0..2] of real = (115,105,70);
     gamma: array[0..2] of real = (25,70,20);
     hues: array[0..3] of integer = (2,10,11,14);


{*****************************************}
{*   Déclaration des variables utilisées   *}
{*****************************************}


var
     GraphDriver,GraphMode,adapt,mode,k,xscale,yscale,xoffset,
          yoffset,pr,index,index2,point,nbre : integer;
     i: longint;
     a,b,c,d,e,f,g,h,m,n,q,r: array[0..3] of real;
     x,y,z,newx,newy,j,ca,cb,cg,sa,sb,sg,alphap,betap,gammap: real;
     p,pk: array[0..3] of real;
     ch1: char;
```

```
{*****************************************************************}
{*  Procédure de calcul d'une image fractale et affichage écran *}
{*****************************************************************}

procedure image_draw;

     var
          px,py: integer;
          vx,vy: real;

     begin
          x := 0;
          y := 0;
          z := 0;
          for i:=1 to point do
          begin
               j := Random;
               if j < p[0] then
                    k := 0;
               if (j > p[0]) and (j < p[1]) then
                    k := 1;
               if (j > p[1]) and (j < p[2]) then
                    k := 2;
               if j > p[2] then
                    k := 3;
               newx := (a[k]* x + b[k] * y + c[k] * z + n[k]);
               newy := (d[k]* x + e[k] * y + f[k] * z + q[k]);
               z := (g[k] * x + h[k] * y + m[k] * z + r[k]);
               x := newx;
               y := newy;
               vx := x*ca + y*cb + z*cg;
               vy := x*sa + y*sb + z*sg;
               px := Round(vx*xscale + xoffset);
               py := Round(vy*yscale + yoffset);
               if (px>=-320)and(px<320)and(py>=-240)and(py<240)
               then
               PutPixel(px+320,175 - ((93*py) div 128),hues[1]);
          end;
     end;

{***************************}
{*   Programme Principal   *}
{***************************}

begin

     {*  Demande à l'utilisateur de donner le nombre d'images   *}
     {*  Attention ce nombre défini les images à calculer entre *}
     {*  chaque point intermédiaire                             *}

        clrscr;
        writeln ('Veuillez entrer le nombre d''images de
                  l''animation.');
        readln (nbre);
```

```
{*  Demande à l'utilisateur de donner le nombre de points  *}
{* à calculer par image                                    *}

   writeln;
   writeln ('Veuillez entrer le nombre de points par image
             (de 1000 … 10000).');
   readln (point);


{*  Initialisation de l'écran graphique pour l'affichage   *}
{* de l'attracteur à l'écran                               *}

GraphDriver := 4;
GraphMode := EGAHi;
InitGraph(graphDriver,GraphMode,'');
setEGApalette(0,8);
setEGApalette(2,2);
setEGApalette(10,58);
setEGApalette(11,62);
setEGApalette(14,26);

{*  Initialisation des variables contenant les paramètres  *}
{*  de l'IFS à calculer (ici une fougère en 3D)            *}

a[0] :=0; a[1] := 0.83; a[2] := 0.22; a[3] := -0.22;
b[0] := 0; b[1] := 0; b[2] := -0.23; b[3] := 0.23;
c[0] := 0; c[1] := 0; c[2] :=0; c[3] := 0;
d[0] := 0; d[1] := 0; d[2] := 0.24; d[3] := 0.24;
e[0] := 0.18; e[1] := 0.86; e[2] := 0.22; e[3] := 0.22;
f[0] := 0; f[1] := 0.1; f[2] := 0; f[3] := 0;
g[0] :=0; g[1] := 0; g[2] := 0; g[3] := 0;
h[0] := 0; h[1] := -0.12; h[2] := 0; h[3] := 0;
m[0] := 0; m[1] := 0.84; m[2] :=0.32; m[3] := 0.32;
n[0] := 0; n[1] := 0; n[2] := 0; n[3] := 0;
q[0] := 0; q[1] := 1.62; q[2] := 0.82; q[3] := 0.82;
r[0] := 0; r[1] := 0; r[2] := 0; r[3] := 0;
p[0] := 0.01; p[1] := 0.85; p[2] := 0.92; p[3] := 1.0;

{*   Variables d'échelle et de centrage de l'image   *}

xscale := 40;
yscale := 50;
xoffset := 60;
yoffset := -180;

SetBkColor(0);
ClearDevice;


{*   Début de la boucle effectuant le tour du nombre de   *}
{*   points intermédiaires                                *}

   for index2:=0 to 1 do
   begin
```

```
      {*   Calcul de l'écart moyen défini entre les IFS des 3   *}
      {*   points intermédiaires et ceci en fonction du nombre  *}
      {*   d'images définies pour l'animation                   *}

           alphap := (alpha[1+index2]-alpha[index2])/nbre;
           betap := (beta[1+index2]-beta[index2])/nbre;
           gammap := (gamma[1+index2]-gamma[index2])/nbre;


      {*   Début de la boucle principale exécutant le calcul    *}
      {*   des images de 1 à nbre et ceci entre chaque point    *}
      {*   intermédiaire défini (3)                             *}

      for index:=1 to nbre do
      begin
                Cleardevice;

         {*   Ajout des écarts moyens pour les angles   *}

           ca := cos((alpha[index2]+index*alphap)*0.0174533);
           cb := cos((beta[index2]+index*betap)*0.0174533);
           cg := cos((gamma[index2]+index*gammap)*0.0174533);
           sa := sin((alpha[index2]+index*alphap)*0.0174533);
           sb := sin((beta[index2]+index*betap)*0.0174533);
           sg := sin((gamma[index2]+index*gammap)*0.0174533);

         {*   Appel de la procédure de calcul d'image   *}

           image_draw;
      end;
         end;
      ch1 := ReadKey;
end.
```

Annexe C : Programme "*FractWin*"

```
program FractWin;


{*****************************************************************}
{*   Ce programme réalise des animations d'images fractales de   *}
{* type IFS et plus particulièrement d'IFS ressemblant à des     *}
{* plantes. Il a été réalisé dans le cadre d'un mémoire en info  *}
{* sur ce sujet. Son utilisation et sa conception sont donc      *}
{* développées dans le texte accompagnant le mémoire ainsi que   *}
{* l'étude du problème.                                          *}
{*                                                               *}
{*   Il serait trop long de détaillé ici l'ensemble des données  *}
{* présentes dans le programme. Les commentaires que vous y      *}
{* trouverez seront donc de type indicatif sur les opérations    *}
{ effectuées par les différentes routines.                       *}
{*****************************************************************}


{*****************************************************************}
{*   Chargement de la ressource contenant les caractéristiques   *}
{*   des écrans, menus et icones                                 *}
{*****************************************************************}

{$R frwin2.res}

{*****************************************}
{*   Chargement des librairies utilisées   *}
{*****************************************}

uses BWCC,WinTypes,WinProcs,OWindows,ODialogs,Strings,OStdDlgs;


{*****************************************************************}
{*   Déclaration des constantes contenant les identificateurs    *}
{* de boutons, items de menu...                                  *}
{*****************************************************************}

const

MaxImage = 5;
id_Menu_Princ = 100;
id_modif_ifs = 105;
id_modif_anim = 106;
id_calcul = 107;
id_avant = 108;
id_arriere = 109;
id_arret = 123;
id_valeur = 200;
id_rb1 = 400;
id_rb2 = 401;
id_rb3 = 402;
id_rb4 = 403;
id_rb5 = 404;
cm_Nouveau = 150;
cm_Ouvrir = 151;
cm_Sauver = 152;
cm_Quitter = 153;
cm_FarnNormal = 160;
cm_Farn3D = 161;
```

```
cm_Arbre = 163;
cm_Aproposde = 180;
cm_AnimReset = 610;
cm_AnimFarn2D = 611;
cm_AnimFarn3D = 612;
cm_AnimFarn3DPG = 613;
cm_AnimFarn3DPD = 614;
cm_AnimArbre = 615;

{*****************************************************************}
{*   Déclaration des variables globales. On y retrouve les       *}
{* tableaux qui contiendront les valeurs d'IFS ainsi que les     *}
{* variables pour les animations (nbre image, ralenti...)        *}
{*****************************************************************}

var

  NbImage, Ralenti, PointInter, NumImage, PtInitX, PtInitY, NbImageMem,
      Palette : Integer;
  DureeAnim, EchelleX, EchelleY, ImageCalcul : real;
  NbPoint : Longint;
  a,b,c,d,e,f,g,h,m,n,q,r,p : array [1..MaxImage,1..4] of real;
  alpha, beta, gamma : array [1..MaxImage,1..1] of real;
  ax,bx,cx,dx,ex,fx,gx,hx,mx,nx,qx,rx,px :array [1..MaxImage,1..4] of real;
  alphax, betax, gammax : array [1..MaxImage,1..1] of real;
  DC: HDC;
  TheBitmap : array [1..200] of HBitmap;
  Color : TColorRef;
  HdMenu, HdMenuGrayed : HMenu;
  Arret : Boolean;


{*****************************}
{*   Déclaration des types   *}
{*****************************}

type

 Angle = array [1..MaxImage,1..1] of real;
 Point = array [1..MaxImage,1..4] of real;
 Edition = array [1..4] of PEdit;


{*************************************************}
{*   Définition des objets principaux du programme   *}
{*************************************************}

 MenuApplication = object(TApplication)
   procedure InitMainWindow; virtual;
 end;

 PDialogTest  = ^TDialogTest;
 TDialogTest  = object (TDialog)
 end;

     {*   Objet définissant la fenêtre fixe qui va contenir   *}
     {*   les bitmaps pour les animations                     *}

 PFenBitmap = ^TFenBitmap;
 TFenBitmap = object (TWindow)
    constructor Init (AParent : PWindowsObject; ATitle : PChar);
    procedure GetWindowClass (var AWndClass: TWndClass); virtual;
    procedure SetupWindow; virtual;
    procedure Paint(PaintDC: HDC; var PaintInfo: TPaintStruct); virtual;
```

```
   procedure CloseWindow; virtual;
end;


    {*   Objet définissant la fenêtre principale du programme   *}

PDialogPrinc = ^TDialogPrinc;
TDialogPrinc = object (TDialog)
   STNbreImage : PStatic;
   STRalenti : PStatic;
   STNbrePoints : PStatic;
   STPointInter : PStatic;
   STDureeAnim : PStatic;
   STEchelleX : PStatic;
   STEchelleY : PStatic;
   STPtInitX : PStatic;
   STPtInitY : PStatic;
   STImageCalcul : PStatic;
   Fen : PFenBitmap;
   constructor Init (AParent: PWindowsObject; AName: PChar);
   procedure CloseWindow; virtual;
   procedure SetupWindow; virtual;
   procedure ButtonAnim (var Msg:TMessage);virtual id_First+id_modif_anim;
   procedure ButtonCalcul (var Msg: TMessage); virtual id_First+id_calcul;
   Function MakeBitmap (index, index2: integer): HBitmap;
   procedure ButtonAvant (var Msg: TMessage); virtual id_First+id_Avant;
   procedure ButtonArriere (var Msg:TMessage);virtual id_First+id_Arriere;
   procedure ButtonArret (var Msg: TMessage); virtual id_First+id_Arret;
   procedure ButtonIfs (var Msg: TMessage); virtual id_First+id_modif_ifs;
end;

PMenuWindow = ^MenuWindow;
MenuWindow = object(TWindow)
   DialPrinc: PDialogPrinc;
   constructor Init(AParent: PWindowsObject; ATitle: PChar);
   procedure SetupWindow; virtual;
   procedure GetWindowClass (var AWndClass: TWndClass); virtual;
   procedure CMNouveau (var Msg: TMessage); virtual cm_First+cm_Nouveau;
   procedure CMOuvrir (var Msg: TMessage); virtual cm_First+cm_Ouvrir;
   procedure CMSauver (var Msg: TMessage); virtual cm_First+cm_Sauver;
   procedure CMQuitter (var Msg: TMessage); virtual cm_First+cm_Quitter;
   procedure CMFarnNormal (var Msg: TMessage); virtual
                                          cm_First+cm_FarnNormal;
   procedure CMFarn3D (var Msg: TMessage); virtual cm_First+cm_Farn3D;
   procedure CMArbre (var Msg: TMessage); virtual cm_First+cm_Arbre;
   procedure CMAproposde (var Msg:TMessage);virtual cm_First+cm_Aproposde;
end;


    {*   Objet définissant la boîte de dialogue de changement de   *}
    {*   l'échelle et du point initial                             *}

PDialogIfs = ^TDialogIfs;
TDialogIfs = object (TDialog)
   PEchelleX, PEchelleY, PPtInitX, PPtInitY : PEdit;
   constructor Init (AParent: PWindowsObject; AName: PChar);
   procedure SetupWindow; virtual;
   procedure OK (var Msg: TMessage); virtual id_First + IDOK;
end;


    {*   Objet définissant la boîte de dialogue de modification   *}
    {*   des paramètres de l'animation                            *}
```

```
 PDialogAnim = ^TDialogAnim;
 TDialogAnim = object (TDialog)
    PNbImage, PNbPoint, PRalenti, PPointInter : PEdit;
    constructor Init (AParent: PWindowsObject; AName: PChar);
    procedure SetupWindow; virtual;
    procedure ButtonValeur (var Msg: TMessage); virtual id_First+id_valeur;
    procedure OK (var Msg: TMessage); virtual id_First+IDOK;
 end;


     {*   Objet définissant la boîte de dialogue de définition des   *}
     {*   valeurs des points intermédiaires                          *}

 PDialogPointInter  = ^TDialogPointInter;
 TDialogPointInter  = object (TDialog)
    Pa, Pb, Pc, Pd, Pe, Pf, Pg, Ph, Pm, Pn, Pq, Pr, Pp : Edition;
    Palpha, Pbeta, Pgamma : PEdit;
    constructor Init (AParent: PWindowsObject; AName: PChar);
    procedure SetupWindow; virtual;
    procedure Ok (var Msg: TMessage); virtual id_First + IDOK;
    procedure Cancel (var Msg: TMessage); virtual id_First + IDCANCEL;
    procedure CMAnimFarnReset (var Msg: TMessage); virtual
                                        cm_First+cm_AnimReset;
    procedure CMAnimFarn2D (var Msg: TMessage); virtual
                                        cm_First+cm_AnimFarn2D;
    procedure CMAnimFarn3D (var Msg: TMessage); virtual
                                        cm_First+cm_AnimFarn3D;
    procedure CMAnimFarn3DPG (var Msg: TMessage); virtual
                                        cm_First+cm_AnimFarn3DPG;
    procedure CMAnimFarn3DPD (var Msg: TMessage); virtual
                                        cm_First+cm_AnimFarn3DPD;
    procedure CMAnimArbre (var Msg: TMessage); virtual
                                        cm_First+cm_AnimArbre;
    procedure RB1 (var Msg: TMessage); virtual id_First + id_rb1;
    procedure RB2 (var Msg: TMessage); virtual id_First + id_rb2;
    procedure RB3 (var Msg: TMessage); virtual id_First + id_rb3;
    procedure RB4 (var Msg: TMessage); virtual id_First + id_rb4;
    procedure RB5 (var Msg: TMessage); virtual id_First + id_rb5;
end;


{*************************************************************}
{*   Procédure de mise à zéro des valeurs contenues dans les   *}
{* tableaux définissant les IFS des points intermédiaires.     *}
{*************************************************************}

procedure ResetValeur;
var
   i, j : Integer;

begin
   for i:= 1 to 5 do
   begin
      alpha[i,1] := 0;
      beta[i,1] := 0;
      gamma[i,1] := 0;
      for j:= 1 to 4 do
      begin
         a[i,j]:=0;
         b[i,j]:=0;
         c[i,j]:=0;
         d[i,j]:=0;
         e[i,j]:=0;
```

```
            f[i,j]:=0;
            g[i,j]:=0;
            h[i,j]:=0;
            m[i,j]:=0;
            n[i,j]:=0;
            q[i,j]:=0;
            r[i,j]:=0;
            p[i,j]:=0;
         end;
      end;
   end;


{*************************************************************}
{*   Procédure de définition d'un IFS fougère au point inter-  *}
{* médiaire spécifié.                                          *}
{*************************************************************}

procedure ValeurPointFarn;
var
   NImage : Integer;
begin
   NImage := NumImage;
   a[NImage,1]:=0; a[NImage,2]:=0.83; a[NImage,3]:=0.22; a[NImage,4]:=0.22;
   b[NImage,1]:=0; b[NImage,2]:=0; b[NImage,3]:=-0.23; b[NImage,4]:=0.23;
   c[NImage,1]:=0; c[NImage,2]:=0; c[NImage,3]:=0; c[NImage,4]:=0;
   d[NImage,1]:=0; d[NImage,2]:=0; d[NImage,3]:=0.24; d[NImage,4]:=0.24;
   e[NImage,1]:=0.18;e[NImage,2]:=0.86;e[NImage,3]:=0.22;e[NImage,4]:=0.22;
   f[NImage,1]:=0; f[NImage,2]:=0.1; f[NImage,3]:=0; f[NImage,4]:=0;
   g[NImage,1]:=0; g[NImage,2]:=0; g[NImage,3]:=0; g[NImage,4]:=0;
   h[NImage,1]:=0; h[NImage,2]:=-0.12; h[NImage,3]:=0; h[NImage,4]:=0;
   m[NImage,1]:=0; m[NImage,2]:=0.84; m[NImage,3]:=0.32; m[NImage,4]:=0.32;
   n[NImage,1]:=0; n[NImage,2]:=0; n[NImage,3]:=0; n[NImage,4]:=0;
   q[NImage,1]:=0; q[NImage,2]:=1.62; q[NImage,3]:=0.82; q[NImage,4]:=0.82;
   r[NImage,1]:=0; r[NImage,2]:=0; r[NImage,3]:=0; r[NImage,4]:=0;
   p[NImage,1]:=0.01;p[NImage,2]:=0.85;p[NImage,3]:=0.92;p[NImage,4]:=1.0;
end;


{*************************************************************}
{*   Initialisation de la fenêtre principale et des variables *}
{* globales définissant les animations par défaut.           *}
{*************************************************************}

procedure MenuApplication.InitMainWindow;
begin
   MainWindow := New(PMenuWindow, Init(nil, 'Animations Fractales'));
   NbImage := 15;
   Ralenti := 1;
   NbPoint := 25000;
   PointInter := 2;
   NumImage := 1;
   DureeAnim := 1.5;
   EchelleX := 100;
   EchelleY := 100;
   PtInitX := 150;
   PtInitY := 300;
   NbImageMem := 0;
   ImageCalcul := 0;
end;

procedure MenuWindow.GetWindowClass (var AWndClass: TWndClass);
begin
   TWindow.GetWindowClass (AWndClass);
```

```
   AWndClass.HIcon := LoadIcon (HInstance, 'ICON_1');
end;


{*************************************************************}
{*   Construction de la fenêtre principale avec définition de *}
{* ces caractéristiques.                                      *}
{*************************************************************}

constructor MenuWindow.Init(AParent: PWindowsObject; ATitle: PChar);
begin
  TWindow.Init(AParent, ATitle);
  DialPrinc := New (PDialogPrinc, Init (@ Self,'PRINC'));
  Attr.Menu := LoadMenu(HInstance, 'MENU_PRINC');
  Attr.X := 0;
  Attr.Y := 0;
  Attr.H := 480;
  Attr.W := 640;
  HdMenu := Attr.Menu;
end;

procedure MenuWindow.SetupWindow;
begin
   TWindow.SetupWindow;
   Application^.MakeWindow (DialPrinc);
   Application^.SetKbHandler (DialPrinc);
   DialPrinc^.EnableKbHandler;
   SetFocus (GetDlgItem (DialPrinc^.HWindow, 107));
end;


{*********************************************************}
{*   Opérations à réaliser lors de l'appel de la commande *}
{* 'Nouveau' dans le sous-menu 'Fichiers'                 *}
{*********************************************************}

procedure MenuWindow.CMNouveau (var Msg: TMessage);
var
   NbImageChar,RalentiChar,NbPointChar,PointInterChar,DureeAnimChar : array
[1..8] of Char;
   NbImageSt,RalentiSt,NbPointSt,PointInterSt,DureeAnimST : String [7];
   EchelleXChar,EchelleYChar,PtInitXChar,PtInitYChar,ImageCalculChar :
array [1..8] of Char;
   EchelleXSt,EchelleYSt,PtInitXSt,PtInitYSt,ImageCalculSt : String [7];

begin
   NumImage := 1;
   ResetValeur;
   NbImage := 15;
   Ralenti := 1;
   PointInter := 2;
   DureeAnim := 1.5;
   NbPoint := 25000;
   EchelleX := 100;
   EchelleY := 100;
   PtInitX := 150;
   PtInitY := 300;
   ImageCalcul := 0;
   Palette := 1;
   Arret := False;
   Str (NbImage, NbImageSt);
   Str (Ralenti, RalentiSt);
   Str (NbPoint, NbPointSt);
   Str (PointInter, PointInterSt);
```

```
   Str (DureeAnim:3:4, DureeAnimSt);
   Str (EchelleX:3:2, EchelleXSt);
   Str (EchelleY:3:2, EchelleYSt);
   Str (PtInitX, PtInitXSt);
   Str (PtInitY, PtInitYSt);
   Str (ImageCalcul:2:2, ImageCalculSt);
   DialPrinc^.STNbreImage^.SetText (StrPCopy (@ NbImageChar, NbImageSt));
   DialPrinc^.STRalenti^.SetText (StrPCopy (@ RalentiChar, RalentiSt));
   DialPrinc^.STNbrePoints^.SetText (StrPCopy (@ NbPointChar, NbPointSt));
   DialPrinc^.STPointInter^.SetText (StrPCopy (@ PointInterChar,
                                          PointInterSt));
   DialPrinc^.STDureeAnim^.SetText (StrPCopy (@ DureeAnimChar,
                                          DureeAnimST));
   DialPrinc^.STEchelleX^.SetText (StrPCopy (@ EchelleXChar, EchelleXSt));
   DialPrinc^.STEchelleY^.SetText (StrPCopy (@ EchellèYChar, EchelleYSt));
   DialPrinc^.STPtInitX^.SetText (StrPCopy (@ PtInitXChar, PtInitXSt));
   DialPrinc^.STPtInitY^.SetText (StrPCopy (@ PtInitYChar, PtInitYSt));
   DialPrinc^.STImageCalcul^.SetText (StrPCopy (@ ImageCalculChar,
                                          ImageCalculSt));
end;


{*********************************************************}
{*   Opérations à réaliser lors de l'appel de la commande *}
{* 'Ouvrir' dans le sous-menu 'Fichiers'                 *}
{*********************************************************}

procedure MenuWindow.CMOuvrir (var Msg: TMessage);
var
   NomFichier,Pbox : array [1..255] of char;
   Fi : Text;
   i,j : integer;
   NbImageChar, RalentiChar, NbPointChar, PointInterChar, DureeAnimChar :
                                          array [1..8] of Char;
   NbImageSt, RalentiSt, NbPointSt, PointInterSt, DureeAnimST : String [7];
   EchelleXChar,EchelleYChar,PtInitXChar,PtInitYChar:array [1..8] of Char;
   EchelleXSt,EchelleYSt,PtInitXSt,PtInitYSt : String [7];
begin
   StrCopy (@ NomFichier, '*.ani');
   if Application^.ExecDialog (New (PFileDialog, Init (@ Self,
            PChar(sd_FileOpen), @ NomFichier)))=id_OK then
   begin
      Assign (Fi, StrPas(@ NomFichier));
      {$I-} Reset (Fi); {$I+}
      If IoResult <> 0
      then
         MessageBox(HWindow, StrPCopy(@ Pbox,'Fichier '+StrPas(@
            NomFichier)+' non trouvé'), 'Erreur !',mb_IconExclamation)
      else
      begin
         Readln (Fi,NbImage);
         Readln (Fi,NbPoint);
         Readln (Fi,PointInter);
         Readln (Fi,EchelleX);
         Readln (Fi,EchelleY);
         Readln (Fi,PtInitX);
         Readln (Fi,PtInitY);
         Readln (Fi, Palette);
         for i:=1 to 5 do
         begin
            for j:=1 to 4 do
            begin
               Readln (Fi,a[i,j]);
               Readln (Fi,b[i,j]);
```

```
                Readln (Fi,c[i,j]);
                Readln (Fi,d[i,j]);
                Readln (Fi,e[i,j]);
                Readln (Fi,f[i,j]);
                Readln (Fi,g[i,j]);
                Readln (Fi,h[i,j]);
                Readln (Fi,m[i,j]);
                Readln (Fi,n[i,j]);
                Readln (Fi,q[i,j]);
                Readln (Fi,r[i,j]);
                Readln (Fi,p[i,j]);
             end;
             Readln (Fi,alpha[i,1]);
             Readln (Fi,beta[i,1]);
             Readln (Fi,gamma[i,1]);
          end;
          Close (Fi);
       end;

       Ralenti := 1;
       DureeAnim := (PointInter-1)*((NbImage/10)+((Ralenti-1)/5));

       Str (NbImage, NbImageSt);
       Str (Ralenti, RalentiSt);
       Str (NbPoint, NbPointSt);
       Str (PointInter, PointInterSt);
       Str (DureeAnim:3:4, DureeAnimSt);
       Str (EchelleX:3:2, EchelleXSt);
       Str (EchelleY:3:2, EchelleYSt);
       Str (PtInitX, PtInitXSt);
       Str (PtInitY, PtInitYSt);
       DialPrinc^.STNbreImage^.SetText (StrPCopy (@ NbImageChar,NbImageSt));
       DialPrinc^.STRalenti^.SetText (StrPCopy (@ RalentiChar,RalentiSt));
       DialPrinc^.STNbrePoints^.SetText (StrPCopy (@NbPointChar,NbPointSt));
       DialPrinc^.STPointInter^.SetText (StrPCopy (@PointInterChar,
                                          PointInterSt));
       DialPrinc^.STDureeAnim^.SetText (StrPCopy (@ DureeAnimChar,
                                          DureeAnimST));
       DialPrinc^.STEchelleX^.SetText (StrPCopy(@ EchelleXChar,EchelleXSt));
       DialPrinc^.STEchelleY^.SetText (StrPCopy(@ EchelleYChar,EchelleYSt));
       DialPrinc^.STPtInitX^.SetText (StrPCopy (@ PtInitXChar, PtInitXSt));
       DialPrinc^.STPtInitY^.SetText (StrPCopy (@ PtInitYChar, PtInitYSt));
    end;
end;


{*********************************************************}
{*   Opérations à réaliser lors de l'appel de la commande *}
{* 'Sauver' dans le sous-menu 'Fichiers'                 *}
{*********************************************************}

procedure MenuWindow.CMSauver (var Msg: TMessage);
var
   NomFichier : array [1..255] of char;
   Fi : Text;
   i,j : integer;
begin
   StrCopy (@ NomFichier, '*.ani');
   If Application^.ExecDialog (New (PFileDialog, Init (@ Self,
PChar(sd_FileOpen), @ NomFichier))) = id_Ok
   then
   begin
      Assign (Fi, StrPas(@ NomFichier));
      {$I-} Rewrite (Fi); {$I+}
```

```
      Writeln (Fi,NbImage);
      Writeln (Fi,NbPoint);
      Writeln (Fi,PointInter);
      Writeln (Fi,EchelleX);
      Writeln (Fi,EchelleY);
      Writeln (Fi,PtInitX);
      Writeln (Fi,PtInitY);
      Writeln (Fi, Palette);
      for i:=1 to 5 do
      begin
         for j:=1 to 4 do
         begin
            Writeln (Fi,a[i,j]);
            Writeln (Fi,b[i,j]);
            Writeln (Fi,c[i,j]);
            Writeln (Fi,d[i,j]);
            Writeln (Fi,e[i,j]);
            Writeln (Fi,f[i,j]);
            Writeln (Fi,g[i,j]);
            Writeln (Fi,h[i,j]);
            Writeln (Fi,m[i,j]);
            Writeln (Fi,n[i,j]);
            Writeln (Fi,q[i,j]);
            Writeln (Fi,r[i,j]);
            Writeln (Fi,p[i,j]);
         end;
         Writeln (Fi,alpha[i,1]);
         Writeln (Fi,beta[i,1]);
         Writeln (Fi,gamma[i,1]);
      end;
      Close (Fi);
   end;
end;


{*********************************************************}
{*   Opérations à réaliser lors de l'appel de la commande *}
{* 'Quitter' dans le sous-menu 'Fichiers'                 *}
{*********************************************************}

procedure MenuWindow.CMQuitter (var Msg: TMessage);
begin
   CloseWindow;
end;


{*********************************************************}
{*   Opérations à réaliser lors de l'appel de la commande *}
{* 'Farn Normal' dans le sous-menu 'Choix IFS'            *}
{*********************************************************}

procedure MenuWindow.CMFarnNormal (var Msg: TMessage);
var
   NbImageChar, RalentiChar, NbPointChar, PointInterChar, DureeAnimChar :
                                          array [1..8] of Char;
   NbImageSt, RalentiSt, NbPointSt, PointInterSt, DureeAnimST : String [7];
   EchelleXChar,EchelleYChar,PtInitXChar,PtInitYChar:array [1..8] of Char;
   EchelleXSt,EchelleYSt,PtInitXSt,PtInitYSt : String [7];

begin
   NumImage := 2;
   ResetValeur;
   ValeurPointFarn;
   alpha[1,1]:=0; beta[1,1]:=80; gamma[1,1]:=60;
```

```
   alpha[2,1]:=0; beta[2,1]:=80; gamma[2,1]:=60;
   m[2,1]:=0; m[2,2]:=0; m[2,3]:=0; m[2,4]:=0;
   p[1,1]:= 0.01; p[1,2]:= 0.85; p[1,3]:= 0.92; p[1,4]:= 1;
   PointInter := 2;
   EchelleX := 65;
   EchelleY := 65;
   PtInitX := 150;
   PtInitY := 300;
   Palette := 1;
   NbPoint := 25000;
   NbImage := 10;
   DureeAnim := 1;

   Str (NbImage, NbImageSt);
   Str (Ralenti, RalentiSt);
   Str (NbPoint, NbPointSt);
   Str (PointInter, PointInterSt);
   Str (DureeAnim:3:4, DureeAnimSt);
   Str (EchelleX:3:2, EchelleXSt);
   Str (EchelleY:3:2, EchelleYSt);
   Str (PtInitX, PtInitXSt);
   Str (PtInitY, PtInitYSt);
   DialPrinc^.STNbreImage^.SetText (StrPCopy (@ NbImageChar, NbImageSt));
   DialPrinc^.STRalenti^.SetText (StrPCopy (@ RalentiChar, RalentiSt));
   DialPrinc^.STNbrePoints^.SetText (StrPCopy (@ NbPointChar, NbPointSt));
   DialPrinc^.STPointInter^.SetText (StrPCopy (@ PointInterChar,
                                        PointInterSt));
   DialPrinc^.STDureeAnim^.SetText (StrPCopy(@ DureeAnimChar,DureeAnimST));
   DialPrinc^.STEchelleX^.SetText (StrPCopy (@ EchelleXChar, EchelleXSt));
   DialPrinc^.STEchelleY^.SetText (StrPCopy (@ EchelleYChar, EchelleYSt));
   DialPrinc^.STPtInitX^.SetText (StrPCopy (@ PtInitXChar, PtInitXSt));
   DialPrinc^.STPtInitY^.SetText (StrPCopy (@ PtInitYChar, PtInitYSt));
 end;


{*********************************************************}
{*   Opérations à réaliser lors de l'appel de la commande *}
{* 'Farn 3D' dans le sous-menu 'Choix IFS'                *}
{*********************************************************}

procedure MenuWindow.CMFarn3D (var Msg: TMessage);
var
   NbImageChar, RalentiChar, NbPointChar, PointInterChar, DureeAnimChar :
                                        array [1..8] of Char;
   NbImageSt, RalentiSt, NbPointSt, PointInterSt, DureeAnimST : String [7];
   EchelleXChar,EchelleYChar,PtInitXChar,PtInitYChar:array [1..8] of Char;
   EchelleXSt,EchelleYSt,PtInitXSt,PtInitYSt . String [7];
   NImage : Integer;

begin
   ResetValeur;
   PointInter := 3;
   for NImage := 1 to 3 do
   begin
      NumImage:=NImage;
      ValeurPointFarn;
   end;

   EchelleX := 90;
   EchelleY := 90;
   PtInitX := 200;
   PtInitY := 300;
   Palette := 1;
   NbPoint := 25000;
```

```
   NbImage := 10;
   DureeAnim := 2;
   alpha[1,1] := 30; alpha[2,1] := 45; alpha[3,1]:= 15;
   beta[1,1] := 115; beta[2,1] := 105; beta[3,1] := 70;
   gamma[1,1] := 25; gamma[2,1] := 70; gamma[3,1] := 20;

   Str (NbImage, NbImageSt);
   Str (Ralenti, RalentiSt);
   Str (NbPoint, NbPointSt);
   Str (PointInter, PointInterSt);
   Str (DureeAnim:3:4, DureeAnimSt);
   Str (EchelleX:3:2, EchelleXSt);
   Str (EchelleY:3:2, EchelleYSt);
   Str (PtInitX, PtInitXSt);
   Str (PtInitY, PtInitYSt);
   DialPrinc^.STNbreImage^.SetText (StrPCopy (@ NbImageChar, NbImageSt));
   DialPrinc^.STRalenti^.SetText (StrPCopy (@ RalentiChar, RalentiSt));
   DialPrinc^.STNbrePoints^.SetText (StrPCopy (@ NbPointChar, NbPointSt));
   DialPrinc^.STPointInter^.SetText (StrPCopy (@ PointInterChar,
                                        PointInterSt));
   DialPrinc^.STDureeAnim^.SetText (StrPCopy(@ DureeAnimChar,DureeAnimST));
   DialPrinc^.STEchelleX^.SetText (StrPCopy (@ EchelleXChar, EchelleXSt));
   DialPrinc^.STEchelleY^.SetText (StrPCopy (@ EchelleYChar, EchelleYSt));
   DialPrinc^.STPtInitX^.SetText (StrPCopy (@ PtInitXChar, PtInitXSt));
   DialPrinc^.STPtInitY^.SetText (StrPCopy (@ PtInitYChar, PtInitYSt));
 end;


{*********************************************************}
{*   Opérations à réaliser lors de l'appel de la commande *}
{* 'Arbre' dans le sous-menu 'Choix IFS'                  *}
{*********************************************************}

procedure MenuWindow.CMArbre (var Msg: TMessage);
var
   NImage : Integer;
   NbImageChar, RalentiChar, NbPointChar, PointInterChar, DureeAnimChar :
                                        array [1..8] of Char;
   NbImageSt, RalentiSt, NbPointSt, PointInterSt, DureeAnimST : String [7];
   EchelleXChar,EchelleYChar,PtInitXChar,PtInitYChar:array [1..8] of Char;
   EchelleXSt,EchelleYSt,PtInitXSt,PtInitYSt : String [7];
begin
   ResetValeur;
   NumImage := 2;
   PointInter := 2;
   EchelleX := 100;
   EchelleY := 100;
   PtInitX := 150;
   PtInitY := 300;
   Palette := 2;
   NbPoint := 20000;
   NbImage := 10;
   DureeAnim := 1;

   NImage := NumImage;
   a[NImage,1]:=0.01;a[NImage,2]:=0.55;a[NImage,3]:=0.7;a[NImage,4]:=0.6;
   b[NImage,1]:=0; b[NImage,2]:=-0.24; b[NImage,3]:=0.2; b[NImage,4]:=0.05;
   d[NImage,1]:=0; d[NImage,2]:=0.24; d[NImage,3]:=-0.2;d[NImage,4]:=-0.05;
   e[NImage,1]:=0.45;e[NImage,2]:=0.65;e[NImage,3]:=0.7;e[NImage,4]:=0.6;
   n[NImage,1]:=0; n[NImage,2]:=0; n[NImage,3]:=0; n[NImage,4]:=0;
   q[NImage,1]:=-0.15;q[NImage,2]:=1.3;q[NImage,3]:=1.3;q[NImage,4]:=-0.15;
   p[NImage,1]:=0.03; p[NImage,2]:=0.5; p[NImage,3]:=1; p[NImage,4]:=1.5;
   p[1,1]:=0.07; p[1,2]:=0.41; p[1,3]:=1; p[1,4]:=1.5;
```

```
   alpha[1,1]:= 0; beta[1,1]:= 90; gamma[1,1]:= 60;
   alpha[2,1]:= 0; beta[2,1]:= 90; gamma[2,1]:= 60;

   Str (NbImage, NbImageSt);
   Str (Ralenti, RalentiSt);
   Str (NbPoint, NbPointSt);
   Str (PointInter, PointInterSt);
   Str (DureeAnim:3:4, DureeAnimSt);
   Str (EchelleX:3:2, EchelleXSt);
   Str (EchelleY:3:2, EchelleYSt);
   Str (PtInitX, PtInitXSt);
   Str (PtInitY, PtInitYSt);
   DialPrinc^.STNbreImage^.SetText (StrPCopy (@ NbImageChar, NbImageSt));
   DialPrinc^.STRalenti^.SetText (StrPCopy (@ RalentiChar, RalentiSt));
   DialPrinc^.STNbrePoints^.SetText (StrPCopy (@ NbPointChar, NbPointSt));
   DialPrinc^.STPointInter^.SetText (StrPCopy (@ PointInterChar,
                                        PointInterSt));
   DialPrinc^.STDureeAnim^.SetText (StrPCopy(@ DureeAnimChar,DureeAnimST));
   DialPrinc^.STEchelleX^.SetText (StrPCopy (@ EchelleXChar, EchelleXSt));
   DialPrinc^.STEchelleY^.SetText (StrPCopy (@ EchelleYChar, EchelleYSt));
   DialPrinc^.STPtInitX^.SetText (StrPCopy (@ PtInitXChar, PtInitXSt));
   DialPrinc^.STPtInitY^.SetText (StrPCopy (@ PtInitYChar, PtInitYSt));
end;


{*********************************************************}
{*   Opérations à réaliser lors de l'appel de la commande *}
{* 'A propos de' dans le sous-menu '?'                     *}
{*********************************************************}

procedure MenuWindow.CMAproposde (var Msg: TMessage);
begin
   Application^.ExecDialog (New (PDialogTest,Init (@ Self, 'A_PROPOS')));
end;


{***********************************************}
{*   Initialisation de la fenêtre principale   *}
{***********************************************}

constructor TDialogPrinc.Init (AParent: PWindowsObject; AName: PChar);
begin
   TDialog.Init (AParent, AName);
   Fen := New (PFenBitmap, Init (@ Self, ''));
   STNbreImage := New (PStatic, InitResource(@ Self, 110, 3));
   STRalenti := New (PStatic, InitResource (@ Self, 111, 2));
   STNbrePoints := New (PStatic, InitResource (@ Self, 112, 5));
   STPointInter := New (PStatic, InitResource (@ Self, 113, 2));
   STDureeAnim := New (PStatic, InitResource (@ Self, 114, 5));
   STEchelleX := New (PStatic, InitResource (@ Self, 116, 6));
   STEchelleY := New (PStatic, InitResource (@ Self, 117, 6));
   STPtInitX := New (PStatic, InitResource (@ Self, 118, 5));
   STPtInitY := New (PStatic, InitResource (@ Self, 119, 5));
   STIMageCalcul := New (PStatic, InitResource (@ Self, 179, 4));
end;


{************************************}
{*   Setup de la fenêtre principale   *}
{************************************}

procedure TDialogPrinc.SetupWindow;
var
   NbImageChar, RalentiChar, NbPointChar, PointInterChar, DureeAnimChar :
```

```
                                        array [1..8] of Char;
   NbImageSt, RalentiSt, NbPointSt, PointInterSt, DureeAnimST : String [7];
   EchelleXChar,EchelleYChar,PtInitXChar,PtInitYChar,ImageCalculChar :
                                        array [1..8] of Char;
   EchelleXSt,EchelleYSt,PtInitXSt,PtInitYSt,IMageCalculSt : String [7];

begin
   Str (NbImage, NbImageSt);
   Str (Ralenti, RalentiSt);
   Str (NbPoint, NbPointSt);
   Str (PointInter, PointInterSt);
   Str (DureeAnim:3:4, DureeAnimSt);
   Str (EchelleX:3:2, EchelleXSt);
   Str (EchelleY:3:2, EchelleYSt);
   Str (PtInitX, PtInitXSt);
   Str (PtInitY, PtInitYSt);
   Str (ImageCalcul:2:2, ImageCalculSt);
   TDialog.SetupWindow;
   MoveWindow (HWindow, 0, 0, 640, 480, false);
   STNbreImage^.SetText (StrPCopy (@ NbImageChar, NbImageSt));
   STRalenti^.SetText (StrPCopy (@ RalentiChar, RalentiSt));
   STNbrePoints^.SetText (StrPCopy (@ NbPointChar, NbPointSt));
   STPointInter^.SetText (StrPCopy (@ PointInterChar, PointInterSt));
   STDureeAnim^.SetText (StrPCopy (@ DureeAnimChar, DureeAnimST));
   STEchelleX^.SetText (StrPCopy (@ EchelleXChar, EchelleXSt));
   STEchelleY^.SetText (StrPCopy (@ EchelleYChar, EchelleYSt));
   STPtInitX^.SetText (StrPCopy (@ PtInitXChar, PtInitXSt));
   STPtInitY^.SetText (StrPCopy (@ PtInitYChar, PtInitYSt));
   STImageCalcul^.SetText (StrPCopy (@ ImageCalculChar, ImageCalculSt));
   SetWindowPos (Fen^.HWindow, 0, 168, 52, 0, 0, SWP_NOSIZE or SWP_NOZORDER
                                    or SWP_NOACTIVATE or SWP_NOREDRAW);
   DC := GetDC(Fen^.HWindow);
end;


{*****************************************************************}
{*   Réponse à l'appel du bouton "Modifications" de la fenêtre  *}
{* principale et opérations effectuées au retour de l'appel     *}
{*****************************************************************}

procedure TDialogPrinc.ButtonAnim (var Msg: TMessage);
var
   NbImageChar,RalentiChar,NbPointChar,PointInterChar,DureeAnimChar : array
                                        [1..8] of Char;
   NbImageSt,RalentiSt,NbPointSt,PointInterSt,DureeAnimSt : String [7];


begin
   Application^.ExecDialog (New (PDialogAnim,Init (@ Self, 'ANIM_PRINC')));
   Str (NbImage, NbImageSt);
   STNbreImage^.SetText (StrPCopy (@ NbImageChar, NbImageSt));
   Str (Ralenti, RalentiSt);
   STRalenti^.SetText (StrPCopy (@ RalentiChar, RalentiSt));
   Str (NbPoint, NbPointSt);
   STNbrePoints^.SetText (StrPCopy (@ NbPointChar, NbPointSt));
   Str (PointInter, PointInterSt);
   STPointInter^.SetText (StrPCopy (@ PointInterChar, PointInterSt));
   Str (DureeAnim:3:3, DureeAnimSt);
   STDureeAnim^.SetText (StrPCopy (@ DureeAnimChar, DureeAnimSt));
   SetFocus (GetDlgItem (HWindow, 107));
end;


{*****************************************************************}
```

```
{*   Réponse à l'appel du bouton "Changements" de la fenêtre  *}
{* principale et opérations effectuées au retour de l'appel   *}
{************************************************************}

procedure TDialogPrinc.ButtonIfs (var Msg: TMessage);
var
   EchelleXChar,EchelleYChar,PtInitXChar,PtInitYChar : array [1..8] of
Char;
   EchelleXSt,EchelleYSt,PtInitXSt,PtInitYSt : String [7];
begin
   Application^.ExecDialog (New (PDialogIfs,Init (@ Self, 'MODIF_IFS')));
   Str (EchelleX:3:2, EchelleXSt);
   STEchelleX^.SetText (StrPCopy (@ EchelleXChar, EchelleXSt));
   Str (EchelleY:3:2, EchelleYSt);
   STEchelleY^.SetText (StrPCopy (@ EchelleYChar, EchelleYSt));
   Str (PtInitX, PtInitXSt);
   STPtInitX^.SetText (StrPCopy (@ PtInitXChar, PtInitXSt));
   Str (PtInitY, PtInitYSt);
   STPtInitY^.SetText (StrPCopy (@ PtInitYChar, PtInitYSt));
   SetFocus (GetDlgItem (HWindow, 107));
end;


{************************************************************}
{*   Fonction effectuant le calcul de l'image de l'attracteur *}
{* de l'IFS et dessin de cette image sur bitmap en mémoire et *}
{* à l'écran.                                                 *}
{************************************************************}

Function TDialogPrinc.MakeBitmap (index, index2: integer): HBitmap;
var
   MemDc : HDC;
   OldBitmap : HBitmap;
   HBits : HBitmap;
   adapt,k,xscale,yscale,xoffset,yoffset,pr,point,s : integer;
   i, ValeurVert,ValeurRouge: longint;
   x,y,z,newx,newy,j,ca,cb,cg,sa,sb,sg,alphaint,betaint,gammaint: real;
   aint,bint,cint,dint,eint,fint,gint,hint,mint,nint,qint,rint,pint :
                                          array [1..4] of real;
   px,py: integer;
   vx,vy: real;


begin
        xscale := Trunc (40*EchelleX/100);
        yscale := Trunc (50*EchelleY/100);
        xoffset := (PtInitX-150);
        yoffset := (130-PtInitY);
        Color := RGB(0,80,0);

             alphaint := (alpha[1+index2,1]-alpha[index2,1])/NbImage;
             betaint := (beta[1+index2,1]-beta[index2,1])/NbImage;
             gammaint := (gamma[1+index2,1]-gamma[index2,1])/NbImage;

                MemDc:= CreateCompatibleDc (DC);
                HBits:= CreateCompatibleBitmap (DC, 300, 300);
                OldBitmap := SelectObject (MemDc, HBits);
                PatBlt (MemDc, 0, 0, 300, 300, whiteness);

            ca := cos((alpha[index2,1]+index*alphaint)*0.0174533);
            cb := cos((beta[index2,1]+index*betaint)*0.0174533);
            cg := cos((gamma[index2,1]+index*gammaint)*0.0174533);
            sa := sin((alpha[index2,1]+index*alphaint)*0.0174533);
            sb := sin((beta[index2,1]+index*betaint)*0.0174533);
```

```
 sg := sin((gamma[index2,1]+index*gammaint)*0.0174533);
for s:=1 to 4 do
begin
   aint[s]:=(a[index2,s]+(index*(a[index2+1,s]-
                  a[index2,s])/(NbImage)));
   bint[s]:=(b[index2,s]+(index*(b[index2+1,s]-
                  b[index2,s])/(NbImage)));
   cint[s]:=(c[index2,s]+(index*(c[index2+1,s]-
                  c[index2,s])/(NbImage)));
   dint[s]:=(d[index2,s]+(index*(d[index2+1,s]-
                  d[index2,s])/(NbImage)));
   eint[s]:=(e[index2,s]+(index*(e[index2+1,s]-
                  e[index2,s])/(NbImage)));
   fint[s]:=(f[index2,s]+(index*(f[index2+1,s]-
                  f[index2,s])/(NbImage)));
   gint[s]:=(g[index2,s]+(index*(g[index2+1,s]-
                  g[index2,s])/(NbImage)));
   hint[s]:=(h[index2,s]+(index*(h[index2+1,s]-
                  h[index2,s])/(NbImage)));
   mint[s]:=(m[index2,s]+(index*(m[index2+1,s]-
                  m[index2,s])/(NbImage)));
   nint[s]:=(n[index2,s]+(index*(n[index2+1,s]-
                  n[index2,s])/(NbImage)));
   qint[s]:=(q[index2,s]+(index*(q[index2+1,s]-
                  q[index2,s])/(NbImage)));
   rint[s]:=(r[index2,s]+(index*(r[index2+1,s]-
                  r[index2,s])/(NbImage)));
   pint[s]:=(p[index2,s]+(index*(p[index2+1,s]-
                  p[index2,s])/(NbImage)));
end;


PatBlt(DC,0,0,300,300,whiteness);
x := 0;
 y := 0;
 z := 0;
 for i:=1 to NbPoint do
 begin
       j := Random;
       if j < pint[1] then
             k := 1;
       if (j > pint[1]) and (j < pint[2]) then
             k := 2;
       if (j > pint[2]) and (j < pint[3]) then
             k := 3;
       if j > pint[3] then
             k := 4;
       newx :=(aint[k]*x + bint[k]*y + cint[k]*z + nint[k]);
       newy :=(dint[k]*x + eint[k]*y + fint[k]*z + qint[k]);
       z := (gint[k]*x + hint[k]*y + mint[k]*z + rint[k]);
       x := newx;
       y := newy;
       vx := x*ca + y*cb + z*cg;
       vy := x*sa + y*sb + z*sg;
       px := Round(vx*xscale + xoffset);
       py := Round(vy*yscale + yoffset);
       if (px>=-150)and(px<150)and(py>=-150)and(py<150) then
             begin
                ValeurVert := (GetGValue(GetPixel (MemDC,
                                          px+149,149 - py)));
                ValeurRouge := (GetRValue(GetPixel (MemDC,
                                          px+149,149 - py)));
                If ((ValeurVert<255) and (ValeurRouge<255)) then
                  If ((ValeurRouge<255)and(ValeurVert>230)) then
```

```
                                Color := RGB((135),(ValeurVert),0) else
                                Color := RGB(ValeurRouge,(ValeurVert+
                                     Trunc(100/(NbPoint/10000))),0) else
                                If Palette = 2 then Color := RGB (135,60,0)
                                               else Color := RGB (0,75,0);
                             SetPixel (DC,px+149,149 - py, Color);
                             SetPixel (MemDc,px+149,149 - py, Color);
                          end;
            end;

           SelectObject (MemDc, OldBitmap);
           DeleteDc (MemDc);
           MakeBitmap := HBits;
end;


{*******************************************************}
{*   Réponse à l'appel du bouton "Calcul" de la fenêtre  *}
{* principale et opérations effectuées au retour         *}
{*******************************************************}

procedure TDialogPrinc.ButtonCalcul (var Msg: TMessage);
var
   index, index2, i : integer;
   EMsg : TMsg;
   ImageCalculChar : array [1..8] of Char;
   ImageCalculSt : String [7];

begin
   Arret := False;
   for i:=1 to NbImageMem do
      DeleteObject (TheBitmap[i]);
   ImageCalcul := 0;
   Str (ImageCalcul:2:2, ImageCalculSt);
   STImageCalcul^.SetText (StrPCopy (@ ImageCalculChar, ImageCalculSt));

   EnableWindow (GetItemHandle(id_Avant), False);
   EnableWindow (GetItemHandle(id_Arriere), False);
   EnableWindow (GetItemHandle(id_Calcul), False);
   EnableWindow (GetItemHandle(id_modif_ifs), False);
   EnableWindow (GetItemHandle(id_modif_anim), False);
   EnableWindow (GetItemHandle(id_Arret), True);
   EnableMenuItem (HdMenu, cm_Nouveau, MF_GRAYED);
   EnableMenuItem (HdMenu, cm_Ouvrir, MF_GRAYED);
   EnableMenuItem (HdMenu, cm_Quitter,MF_GRAYED);
   EnableMenuItem (HdMenu, cm_Sauver, MF_GRAYED);
   EnableMenuItem (HdMenu, cm_Arbre, MF_GRAYED);
   EnableMenuItem (HdMenu, cm_FarnNormal, MF_GRAYED);
   EnableMenuItem (HdMenu, cm_Farn3D, MF_GRAYED);
   EnableMenuItem (HdMenu, cm_Aproposde, MF_GRAYED);

   index2 := 1;
   index := 1;
   While ((index2 < PointInter) and (not (Arret)))do
   begin
      While ((index < NbImage+1) and (not (Arret))) do
      begin
         NbImageMem := (((index2-1)*NbImage)+index);
         TheBitmap[(((index2-1)*NbImage)+index)]:=
                                        MakeBitmap(index,index2);
         ImageCalcul:=100*((index2-1)*NbImage+index)/((PointInter-1)
                                                          *NbImage);
         Str (ImageCalcul:2:2, ImageCalculSt);
         STImageCalcul^.SetText(StrPCopy(@ ImageCalculChar,ImageCalculSt));
```

```
         while (PeekMessage (EMsg, 0, 0, 0, PM_REMOVE)) do
         begin
            TranslateMessage (EMsg);
            DispatchMessage (EMsg);
         end;
         index := index + 1;
      end;
      index2 := index2 + 1;
      index := 1;
   end;
   EnableWindow (GetItemHandle(id_Avant), True);
   EnableWindow (GetItemHandle(id_Arriere), True);
   EnableWindow (GetItemHandle(id_Calcul), True);
   EnableWindow (GetItemHandle(id_modif_ifs), True);
   EnableWindow (GetItemHandle(id_modif_anim), True);
   EnableWindow (GetItemHandle(id_Arret), False);
   EnableMenuItem (HdMenu, cm_Nouveau, MF_ENABLED);
   EnableMenuItem (HdMenu, cm_Ouvrir, MF_ENABLED);
   EnableMenuItem (HdMenu, cm_Quitter,MF_ENABLED);
   EnableMenuItem (HdMenu, cm_Sauver, MF_ENABLED);
   EnableMenuItem (HdMenu, cm_Arbre, MF_ENABLED);
   EnableMenuItem (HdMenu, cm_FarnNormal, MF_ENABLED);
   EnableMenuItem (HdMenu, cm_Farn3D, MF_ENABLED);
   EnableMenuItem (HdMenu, cm_Aproposde, MF_ENABLED);
   EnableKbHandler;
   SetFocus (GetItemHandle (id_calcul));
end;


{*********************************************************}
{*   Réponse à l'appel du bouton "Avant" de la fenêtre  *}
{* principale                                            *}
{*********************************************************}

procedure TDialogPrinc.ButtonAvant (var Msg: TMessage);
var
   MemDc: HDC;
   OldBitmap: HBitmap;
   i,j : integer;
   ral : Longint;

begin
   for i:=1 to (NbImage*(PointInter-1)) do
   begin
      for ral:=1 to ((Ralenti-1)*50000) do
      begin
      end;
      MemDc:= CreateCompatibleDC (DC);
      OldBitmap := SelectObject(MemDc, TheBitmap[i]);
      BitBlt (DC, 0, 0, 300, 300, MemDc, 0, 0, srcCopy);
      SelectObject (MemDc, OldBitmap);
      DeleteDc (MemDc);
   end;
end;


{*********************************************************}
{*   Réponse à l'appel du bouton "Arrière" de la fenêtre  *}
{* principale                                            *}
{*********************************************************}

procedure TDialogPrinc.ButtonArriere (var Msg: TMessage);
var
   MemDc: HDC;
```

```
   OldBitmap: HBitmap;
   i,j : integer;
   ral : Longint;

begin
   for i:=(NbImage*(PointInter-1)) downto 1 do
   begin
      for ral:=1 to ((Ralenti-1)*50000) do
      begin
      end;
      MemDc:= CreateCompatibleDC (DC);
      OldBitmap := SelectObject(MemDc, TheBitmap[i]);
      BitBlt (DC, 0, 0, 300, 300, MemDc, 0, 0, srcCopy);
      SelectObject (MemDc, OldBitmap);
      DeleteDc (MemDc);
   end;
end;


{*****************************************************************}
{*   Réponse à l'appel du bouton "Arrêt Calcul" de la fenêtre    *}
{* principale                                                    *}
{*****************************************************************}

procedure TDialogPrinc.ButtonArret (var Msg: TMessage);
begin
   Arret := True;
end;


{*****************************************************************}
{*   Fermeture de l'application et opérations de "nettoyage"     *}
{*****************************************************************}


procedure TDialogPrinc.CloseWindow;
begin
   ReleaseDC (HWindow, DC);
   TDialog.CloseWindow;
end;


{*****************************************************************}
{*   Constructeur de l'objet Fenêtre fixe contenant les bitmaps *}
{*****************************************************************}

constructor TFenBitmap.Init (AParent : PWindowsObject; ATitle : PChar);
begin
   TWindow.Init (AParent, ATitle);
   Attr.Style := Attr.Style or WS_Child or WS_Visible;
end;

procedure TFenBitmap.GetWindowClass (var AWndClass: TWndClass);
begin
   TWindow.GetWindowClass (AWndClass);
   AWndClass.Style := CS_BYTEALIGNWINDOW;
end;

{*******************************************************}
{*   Setup de l'objet Fenêtre fixe contenant les bitmaps *}
{*******************************************************}

procedure TFenBitmap.SetupWindow;
begin
```

```
   TWindow.SetupWindow;
   SetWindowPos (HWindow, 0, 0, 0, 300, 300, (SWP_NOMOVE or SWP_NOZORDER or
                                                SWP_NOACTIVATE));
end;

{*****************************************************************}
{*   Fonction Paint de l'objet Fenêtre fixe contenant les bitmaps *}
{*****************************************************************}

procedure TFenBitmap.Paint (PaintDC: HDC; var PaintInfo: TPaintStruct);
var
   MemDc: HDC;
   OldBitmap: HBitmap;

begin
   PatBlt(DC,0,0,300,300,whiteness);
   MemDc := CreateCompatibleDC (PaintDC);
   if (NbImageMem > 0) then
      OldBitmap := SelectObject(MemDc, TheBitmap[NbImageMem]);
   BitBlt (DC, 0, 0, 300, 300, MemDc, 0, 0, srcCopy);
   SelectObject (MemDc, OldBitmap);
   DeleteDc (MemDc);
end;

{*************************************************************}
{*   Fermeture de l'objet Fenêtre fixe contenant les bitmaps *}
{*************************************************************}

procedure TFenBitmap.CloseWindow;
begin
   ReleaseDc (HWindow, DC);
   TWindow.CloseWindow;
end;


{*************************************************************}
{*   Constructeur de l'objet concernant la boîte de dialogue  *}
{* de définition des valeurs des IFS intermédiaires           *}
{*************************************************************}

constructor TDialogPointInter.Init (AParent: PWindowsObject; AName: PChar);
var
   i,k : integer;
begin
   TDialog.Init (AParent, AName);
   for i:= 1 to 4 do
   begin
      Pa[i] := New (PEdit, InitResource (@ Self, 300+i, 6));
      Pb[i] := New (PEdit, InitResource (@ Self, 304+i, 6));
      Pc[i] := New (PEdit, InitResource (@ Self, 308+i, 6));
      Pd[i] := New (PEdit, InitResource (@ Self, 312+i, 6));
      Pe[i] := New (PEdit, InitResource (@ Self, 316+i, 6));
      Pf[i] := New (PEdit, InitResource (@ Self, 320+i, 6));
      Pg[i] := New (PEdit, InitResource (@ Self, 324+i, 6));
      Ph[i] := New (PEdit, InitResource (@ Self, 328+i, 6));
      Pm[i] := New (PEdit, InitResource (@ Self, 332+i, 6));
      Pn[i] := New (PEdit, InitResource (@ Self, 336+i, 6));
      Pq[i] := New (PEdit, InitResource (@ Self, 340+i, 6));
      Pr[i] := New (PEdit, InitResource (@ Self, 344+i, 6));
      Pp[i] := New (PEdit, InitResource (@ Self, 348+i, 6));
   end;
   Palpha := New (PEdit, InitResource (@ Self, 360, 7));
   Pbeta := New (PEdit, InitResource (@ Self, 361, 7));
   Pgamma := New (PEdit, InitResource (@ Self, 362, 7));
```

```
end;


{*****************************************************************}
{*   Procédure permettant de réaliser l'affichage d'une valeur   *}
{* à un endroit déterminé de la boîte de dialogue et avec une    *}
{* valeur donnée initialement                                    *}
{*****************************************************************}

procedure affichage (var Calcul: real; Result: PEdit);

   var
      ASt : String [7];
      AChar : array [1..7] of Char;

   begin
      Str (Calcul:6:3, ASt);
      Result^.SetText (StrPCopy (@ AChar, Ast));
   end;


{*****************************************************************}
{*   Procédure permettant de réaliser l'affichage de l'ensemble  *}
{* des valeurs de l'IFS pour un point intermédiaire défini       *}
{*****************************************************************}

procedure AffichageEcranAnim (var
P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13: Edition;
                              var Palpha,Pbeta,Pgamma: PEdit);
   var
      i : integer;
  begin
   for i:= 1 to 4 do
   begin
      affichage (a[NumImage,i], P1[i]);
      affichage (b[NumImage,i], P2[i]);
      affichage (c[NumImage,i], P3[i]);
      affichage (d[NumImage,i], P4[i]);
      affichage (e[NumImage,i], P5[i]);
      affichage (f[NumImage,i], P6[i]);
      affichage (g[NumImage,i], P7[i]);
      affichage (h[NumImage,i], P8[i]);
      affichage (m[NumImage,i], P9[i]);
      affichage (n[NumImage,i], P10[i]);
      affichage (q[NumImage,i], P11[i]);
      affichage (r[NumImage,i], P12[i]);
      affichage (p[NumImage,i], P13[i]);
   end;
   affichage (alpha[NumImage,1], Palpha);
   affichage (beta[NumImage,1], Pbeta);
   affichage (gamma[NumImage,1], Pgamma);
end;


{*****************************************************************}
{*   Procédure permettant de réaliser la récuparation des        *}
{* valeurs de l'IFS pour un point intermédiaire défini           *}
{*****************************************************************}

procedure RecupEcranAnim (var P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13:
                          Edition; var Palpha,Pbeta,Pgamma: PEdit);

   var
      i : integer;
```

```
   procedure recupval (var ACalcul : real; PaCalcul : PEdit);

      var
         code : integer;
         Ast : array [1..7] of Char;

      begin
         PaCalcul^.GetText (@ Ast, 6);
         Val (StrPas (@ Ast), ACalcul, code);
      end;

begin
   for i:= 1 to 4 do
   begin
      recupval (a[NumImage,i], P1[i]);
      recupval (b[NumImage,i], P2[i]);
      recupval (c[NumImage,i], P3[i]);
      recupval (d[NumImage,i], P4[i]);
      recupval (e[NumImage,i], P5[i]);
      recupval (f[NumImage,i], P6[i]);
      recupval (g[NumImage,i], P7[i]);
      recupval (h[NumImage,i], P8[i]);
      recupval (m[NumImage,i], P9[i]);
      recupval (n[NumImage,i], P10[i]);
      recupval (q[NumImage,i], P11[i]);
      recupval (r[NumImage,i], P12[i]);
      recupval (p[NumImage,i], P13[i]);
   end;
   recupval (alpha[NumImage,1], Palpha);
   recupval (beta[NumImage,1], Pbeta);
   recupval (gamma[NumImage,1], Pgamma);
end;


{*****************************************************************}
{*   Setup de l^'objet concernant la boîte de dialogue pour la   *}
{* définition des valeurs des IFS aux points intermédiaires      *}
{*****************************************************************}

procedure TDialogPointInter.SetupWindow;
var
   i : integer;
begin
   TDialog.SetupWindow;
   AffichageEcranAnim(Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                          Palpha,Pbeta,Pgamma);
   SendDlgItemMsg (id_rb1+(NumImage-1),bm_setCheck,1,0);
   for i:=1 to PointInter do
      EnableWindow (GetItemHandle(id_rb1+(i-1)), True);

end;

{*****************************************************************}
{*   Réponse au bouton "OK" dans la boîte de dialogue de définition *}
{* des valeurs des IFS intermédiaires                            *}
{*****************************************************************}

procedure TDialogPointInter.Ok (var Msg: TMessage);
begin
   RecupEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                          Palpha,Pbeta,Pgamma);
   TDialog.Ok (Msg);
end;
```

```
{*************************************************************}
{*   Réponse au bouton "Cancel" dans la boîte de dialogue de  *}
{* définition des valeurs des IFS intermédiaires              *}
{*************************************************************}

procedure TDialogPointInter.Cancel (var Msg: TMessage);
var
   i,j : integer;

begin
   for i:=1 to 5 do
   begin
      alpha[i,1]:=alphax[i,1]; beta[i,1]:=betax[i,1]; gamma[i,1]:=
                                                        gammax[i,1];
      for j:=1 to 4 do
      begin
         a[i,j]:=ax[i,j];b[i,j]:=bx[i,j];c[i,j]:=cx[i,j];d[i,j]:=dx[i,j];
         e[i,j]:=ex[i,j];f[i,j]:=fx[i,j];g[i,j]:=gx[i,j];h[i,j]:=hx[i,j];
         m[i,j]:=mx[i,j];n[i,j]:=nx[i,j];q[i,j]:=qx[i,j];r[i,j]:=rx[i,j];
         p[i,j]:= px[i,j];
      end;
   end;
   TDialog.Cancel(Msg);
end;

{*************************************************************}
{*   Réponse au radio-bouton "Image 1" dans la boîte de dialogue  *}
{* de définition des valeurs des IFS intermédiaires           *}
{*************************************************************}

procedure TDialogPointInter.RB1 (var Msg : TMessage);
begin
   RecupEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                   Palpha,Pbeta,Pgamma);
   NumImage := 1;
   AffichageEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                   Palpha,Pbeta,Pgamma);
end;

{*************************************************************}
{*   Réponse au radio-bouton "Image 2" dans la boîte de dialogue  *}
{* de définition des valeurs des IFS intermédiaires           *}
{*************************************************************}

procedure TDialogPointInter.RB2 (var Msg : TMessage);
begin
   RecupEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                   Palpha,Pbeta,Pgamma);
   NumImage := 2;
   AffichageEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                   Palpha,Pbeta,Pgamma);
end;

{*************************************************************}
{*   Réponse au radio-bouton "Image 3" dans la boîte de dialogue  *}
{* de définition des valeurs des IFS intermédiaires           *}
{*************************************************************}

procedure TDialogPointInter.RB3 (var Msg : TMessage);
begin
   RecupEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                   Palpha,Pbeta,Pgamma);
   NumImage := 3;
```

```
   AffichageEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                    Palpha,Pbeta,Pgamma);
end;

{*****************************************************************}
{*   Réponse au radio-bouton "Image 4" dans la boîte de dialogue  *}
{* de définition des valeurs des IFS intermédiaires               *}
{*****************************************************************}

procedure TDialogPointInter.RB4 (var Msg : TMessage);
begin
   RecupEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                    Palpha,Pbeta,Pgamma);
   NumImage := 4;
   AffichageEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                    Palpha,Pbeta,Pgamma);
end;

{*****************************************************************}
{*   Réponse au radio-bouton "Image 5" dans la boîte de dialogue  *}
{* de définition des valeurs des IFS intermédiaires               *}
{*****************************************************************}

procedure TDialogPointInter.RB5 (var Msg : TMessage);
begin
   RecupEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                    Palpha,Pbeta,Pgamma);
   NumImage := 5;
   AffichageEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                    Palpha,Pbeta,Pgamma);
end;


{*****************************************************************}
{*   Réponse à la commande 'Mise à zéro' du sous-menu 'Type de   *}
{* fractales' dans la boîte de dialogue Point Intermédiaire      *}
{*****************************************************************}

procedure TDialogPointInter.CMAnimFarnReset (var Msg: TMessage);
var
   i : integer;
begin
   for i:=1 to 4 do
   begin
      a[NumImage,i]:=0;b[NumImage,i]:=0;c[NumImage,i]:=0;d[NumImage,i]:=0;
      e[NumImage,i]:=0;f[NumImage,i]:=0;g[NumImage,i]:=0;h[NumImage,i]:=0;
      m[NumImage,i]:=0;n[NumImage,i]:=0;q[NumImage,i]:=0;r[NumImage,i]:=0;
      p[NumImage,i]:=0;
   end;

   alpha[NumImage,1]:=0; beta[NumImage,1]:=0; gamma[NumImage,1]:=0;
   AffichageEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                    Palpha,Pbeta,Pgamma);
end;

{*****************************************************************}
{*   Réponse à la commande 'Farn2D' du sous-menu 'Type de   *}
{* fractales' dans la boîte de dialogue Point Intermédiaire *}
{*****************************************************************}

procedure TDialogPointInter.CMAnimFarn2D (var Msg: TMessage);
begin
   ValeurPointFarn;
   Palette := 1;
```

```
   alpha[NumImage,1]:=0; beta[NumImage,1]:=80; gamma[NumImage,1]:=60;
   m[NumImage,1]:=0; m[NumImage,2]:=0; m[NumImage,3]:=0; m[NumImage,4]:=0;
   AffichageEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                    Palpha,Pbeta,Pgamma);
end;

{*****************************************************************}
{*   Réponse à la commande 'Farn3D Normal' du sous-menu 'Type de   *}
{* fractales' dans la boîte de dialogue Point Intermédiaire       *}
{*****************************************************************}

procedure TDialogPointInter.CMAnimFarn3D (var Msg: TMessage);
begin
   ValeurPointFarn;
   Palette := 1;
   alpha[NumImage,1]:=45; beta[NumImage,1]:=105; gamma[NumImage,1]:=70;
   AffichageEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                    Palpha,Pbeta,Pgamma);
end;

{*****************************************************************}
{*   Réponse à la commande 'Farn3D Plié Gauche' du sous-menu 'Type *}
{* de fractales' dans la boîte de dialogue Point Intermédiaire    *}
{*****************************************************************}

procedure TDialogPointInter.CMAnimFarn3DPG (var Msg: TMessage);
begin
   ValeurPointFarn;
   Palette := 1;
   alpha[NumImage,1]:=30; beta[NumImage,1]:=115; gamma[NumImage,1]:=25;
   AffichageEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                    Palpha,Pbeta,Pgamma);
end;

{*****************************************************************}
{*   Réponse à la commande 'Farn3D Plié Droite' du sous-menu 'Type *}
{* de fractales' dans la boîte de dialogue Point Intermédiaire    *}
{*****************************************************************}

procedure TDialogPointInter.CMAnimFarn3DPD (var Msg: TMessage);
begin
   ValeurPointFarn;
   Palette := 1;
   alpha[NumImage,1]:=15; beta[NumImage,1]:=70; gamma[NumImage,1]:=20;
   AffichageEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                    Palpha,Pbeta,Pgamma);
end;

{*****************************************************************}
{*   Réponse à la commande 'Arbre' du sous-menu 'Type de    *}
{* fractales' dans la boîte de dialogue Point Intermédiaire *}
{*****************************************************************}

procedure TDialogPointInter.CMAnimArbre (var Msg: TMessage);
begin
   a[NumImage,1]:= 0.01; a[NumImage,2]:= 0.55;
   a[NumImage,3]:= 0.7; a[NumImage,4]:= 0.6;
   b[NumImage,1]:= 0; b[NumImage,2]:= -0.24;
   b[NumImage,3]:= 0.2; b[NumImage,4]:= 0.05;
   c[NumImage,1]:= 0; c[NumImage,2]:= 0;
   c[NumImage,3]:= 0; c[NumImage,4]:= 0;
   d[NumImage,1]:= 0; d[NumImage,2]:= 0.24;
   d[NumImage,3]:= -0.2; d[NumImage,4]:= -0.05;
   e[NumImage,1]:= 0.45; e[NumImage,2]:= 0.65;
```

```
   e[NumImage,3]:= 0.7; e[NumImage,4]:= 0.6;
   f[NumImage,1]:= 0; f[NumImage,2]:= 0;
   f[NumImage,3]:= 0; f[NumImage,4]:= 0;
   g[NumImage,1]:= 0; g[NumImage,2]:= 0;
   g[NumImage,3]:= 0; g[NumImage,4]:= 0;
   m[NumImage,1]:= 0; m[NumImage,2]:= 0;
   m[NumImage,3]:= 0; m[NumImage,4]:= 0;
   n[NumImage,1]:= 0; n[NumImage,2]:= 0;
   n[NumImage,3]:= 0; n[NumImage,4]:= 0;
   q[NumImage,1]:= -0.15; q[NumImage,2]:= 1.3;
   q[NumImage,3]:= 1.3; q[NumImage,4]:= -0.15;
   r[NumImage,1]:= 0; r[NumImage,2]:= 0;
   r[NumImage,3]:= 0; r[NumImage,4]:= 0;
   p[NumImage,1]:= 0.03; p[NumImage,2]:= 0.5;
   p[NumImage,3]:= 1; p[NumImage,4]:= 1.5;
   alpha[NumImage,1]:= 0; beta[NumImage,1]:= 90; gamma[NumImage,1]:= 60;
   Palette := 2;
   AffichageEcranAnim (Pa,Pb,Pc,Pd,Pe,Pf,Pg,Ph,Pm,Pn,Pq,Pr,Pp,
                                                    Palpha,Pbeta,Pgamma);
end;


{*****************************************************************}
{*   Constructeur de l'objet concernant la boîte de dialogue de   *}
{* changements des paramètres de l'IFS                           *}
{*****************************************************************}

constructor TDialogIfs.Init (AParent: PWindowsObject; AName: PChar);
begin
   TDialog.Init (AParent, AName);
   PEchelleX := New (PEdit, InitResource (@ Self, 801, 6));
   PEchelleY := New (PEdit, InitResource (@ Self, 802, 6));
   PPtInitX := New (PEdit, InitResource (@ Self, 803, 4));
   PPtInitY := New (PEdit, InitResource (@ Self, 804, 4));
end;

{*********************************************************}
{*   Setup de l'objet concernant la boîte de dialogue de   *}
{* changements des paramètres de l'IFS                     *}
{*********************************************************}

procedure TDialogIfs.SetupWindow;
var
   ASt : String [7];
   AChar : array [1..8] of Char;

begin
   TDialog.SetupWindow;
   Str (EchelleX:3:2, ASt);
   PEchelleX^.SetText (StrPCopy (@ AChar, Ast));
   Str (EchelleY:3:2, ASt);
   PEchelleY^.SetText (StrPCopy (@ AChar, Ast));
   Str (PtInitX, ASt);
   PPtInitX^.SetText (StrPCopy (@ AChar, Ast));
   Str (PtInitY, ASt);
   PPtInitY^.SetText (StrPCopy (@ AChar, Ast));
end;


{*************************************************************}
{*   Réponse à l'enfoncement du bouton 'OK' dans la boîte de  *}
{* dialogue des paramètres de l'IFS                          *}
{*************************************************************}
```

```
procedure TDialogIfs.Ok (var Msg: TMessage);
   var
      code : integer;
      VSt : array [1..8] of Char;

   begin
      PEchelleX^.GetText (@ VSt, 6);
      Val (StrPas (@ VSt), EchelleX, code);
      PEchelleY^.GetText (@ VSt, 6);
      Val (StrPas (@ VSt), EchelleY, code);
      PPtInitX^.GetText (@ VSt, 4);
      Val (StrPas (@ VSt), PtInitX, code);
      PPtInitY^.GetText (@ VSt, 4);
      Val (StrPas (@ VSt), PtInitY, code);
      if (PtInitX > 300) then PtInitX:= 300;
      if (PtInitX < 1) then PtInitX:= 0;
      if (PtInitY > 300) then PtInitY:= 300;
      if (PtInitY < 1) then PtInitY:= 0;
      if (EchelleX > 99) then EchelleX:= 100;
      if (EchelleX < 1) then EchelleX:= 1;
      if (EchelleY > 99) then EchelleY:= 100;
      if (EchelleY < 1) then EchelleY:= 1;
      TDialog.Ok (Msg);

   end;


{*****************************************************************}
{*   Constructeur de l'objet concernant la boîte de dialogue de   *}
{* modifications des paramètres de l'animation                     *}
{*****************************************************************}

constructor TDialogAnim.Init (AParent: PWindowsObject; AName: PChar);
begin
   TDialog.Init (AParent, AName);
   PNbImage := New (PEdit, InitResource (@ Self, 201, 3));
   PNbPoint := New (PEdit, InitResource (@ Self, 202, 6));
   PRalenti := New (PEdit, InitResource (@ Self, 203, 3));
   PPointInter := New (PEdit, InitResource (@ Self, 204, 2));
end;

{*********************************************************}
{*   Setup de l'objet concernant la boîte de dialogue de   *}
{* modifications des paramètres de l'animation             *}
{*********************************************************}

procedure TDialogAnim.SetupWindow;
var
   ASt : String [7];
   AChar : array [1..8] of Char;

begin
   TDialog.SetupWindow;
   Str (NbImage, ASt);
   PNbImage^.SetText (StrPCopy (@ AChar, Ast));
   Str (NbPoint, ASt);
   PNbPoint^.SetText (StrPCopy (@ AChar, Ast));
   Str (Ralenti, ASt);
   PRalenti^.SetText (StrPCopy (@ AChar, Ast));
   Str (PointInter, ASt);
   PPointInter^.SetText (StrPCopy (@ AChar, Ast));
end;
```

```
{*****************************************************************}
{*   Réponse à l'appel de la boîte de dialogue de définition     *}
{* des paramètres des IFS intermédiaires suite en l'enfoncement *}
{* du bouton 'Valeurs' dans la boîte de dialogue de l'animation *}
{*****************************************************************}

procedure TDialogAnim.ButtonValeur (var Msg: TMessage);
var
   code : integer;
   VSt : array [1..7] of Char;
   i,j : integer;

begin
   PPointInter^.GetText (@ VSt, 6);
   Val (StrPas (@ VSt), PointInter, code);
   if (PointInter > 5) then PointInter:= 5;
   if (PointInter < 2) then PointInter:= 2;
   if (NumImage > PointInter) then NumImage:=PointInter;

   for i:=1 to 5 do
   begin
      alphax[i,1]:=alpha[i,1];betax[i,1]:=beta[i,1];
      gammax[i,1]:=gamma[i,1];
      for j:=1 to 4 do
      begin
         ax[i,j]:=a[i,j];bx[i,j]:=b[i,j];cx[i,j]:=c[i,j];dx[i,j]:=d[i,j];
         ex[i,j]:=e[i,j];fx[i,j]:=f[i,j];gx[i,j]:=g[i,j];hx[i,j]:=h[i,j];
         mx[i,j]:=m[i,j]; x[i,j]:= [i,j];qx[i,j]:=q[i,j];rx[i,j]:=r[i,j];
         px[i,j]:=p[i,j];
      end;
   end;
   Application^.ExecDialog (New (PDialogPointInter,Init (@ Self,
                                                    'ANIM_POINTS')));
end;


{*****************************************************************}
{*   Réponse à l'enfoncement du bouton 'OK' dans la boîte de   *}
{* dialogue des paramètres de l'animation                        *}
{*****************************************************************}

procedure TDialogAnim.Ok (var Msg: TMessage);
   var
      code : integer;
      VSt : array [1..7] of Char;

   begin
      PNbImage^.GetText (@ VSt, 6);
      Val (StrPas (@ VSt), NbImage, code);
      PNbPoint^.GetText (@ VSt, 6);
      Val (StrPas (@ VSt), NbPoint, code);
      PRalenti^.GetText (@ VSt, 6);
      Val (StrPas (@ VSt), Ralenti, code);
      PPointInter^.GetText (@ VSt, 6);
      Val (StrPas (@ VSt), PointInter, code);
      if (PointInter > 5) then PointInter:= 5;
      if (PointInter < 2) then PointInter:= 2;
      if (NumImage > PointInter) then NumImage:=PointInter;
      TDialog.Ok (Msg);

      If Ralenti >0 then DureeAnim := (PointInter-1)*((NbImage/10)+
                                                    ((Ralenti-1)/5))
                    else
                    begin
```

```
                        DureeAnim := (PointInter-1)*(NbImage/10);
                        Ralenti := 1;
                    end;
    end;


var

  FractalAnim: MenuApplication;


{*********************************************}
{*   Boucle principale du programme Windows   *}
{*********************************************}

begin
  FractalAnim.Init('Menu');
  FractalAnim.Run;
  FractalAnim.Done;
end.
```

PROGRAMMES

La disquette ci-dessous contient donc les listings précédents ainsi que les programmes exécutables correspondants. Ils sont répartis dans deux répertoires "*Dos*" et "*Windows*". Le premier répertoire contient les deux programmes sous Dos ainsi que les drivers écran nécessaires. Le second contient le programme sous Windows avec un fichier de données contenant les valeurs d'une animation sur un arbre. Ici, en plus du listing, vous trouverez le fichier de ressource contenant les caractéristiques des écrans et menus sous Windows.

Le lancement de ces programmes est simple. En ce qui concerne les programmes sous Dos, il vous suffit de lancer l'exécution à partir du répertoire de la disquette ou encore de recopier l'ensemble de ce répertoire sur disque dur dans le répertoire de votre choix et d'effectuer l'exécution à partir de là. Le programme sous Windows peut également être exécuté sur disquette ou sur disque dur. Je conseillerais cependant l'utilisation du disque dur pour obtenir une rapidité de réponse maximale. Il vous suffit donc de recopier l'ensemble du répertoire "*Windows*" sur disque. Ensuite vous exécutez "*Fractwin*" par l'intermédiaire du "*Gestionnaire de Fichiers*" ou en choisissant la commande "*Exécuter*" du "*Gestionnaire principal*" et en introduisant le chemin d'accès et le nom du programme ensuite. Bon amusement...

Remarque : Si votre Windows ne possède pas le fichier "*BWCC.DLL*" dans le répertoire "*System*" du Windows, il vous faudra recopier ce fichier de la disquette (répertoire "*Win*") sur votre disque dur dans le répertoire "Windows\System".